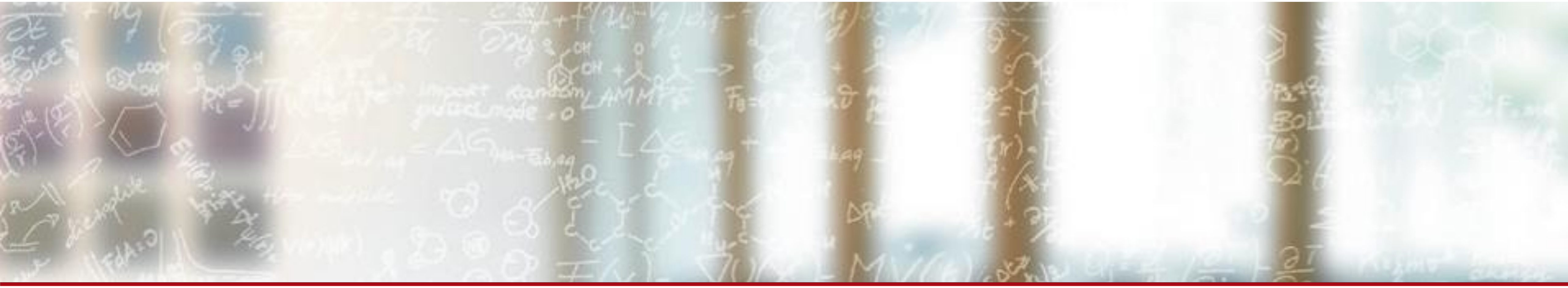
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



In Situ Analysis and Visualization with Catalyst and Ascent

Jean M. Favre,
Senior Visualization Software Engineer
CSCS

May 10, 2022

Thank to different teams

- Thanks to the many actors in the visualization and in-situ dev teams:
- Kitware (François Mazen will join me for the tutorial at ISC22)
- Berkeley Lab, ANL, LLNL, Los Alamos, Sandia, and many others (in particular Cyrus Harrison)

Agenda B.C.

13:00 Welcome, Overview and Motivation

Agenda and technical details for demonstrations

13:15 Introduction to in-situ visualization, workflows and terminology

13:30 Conduit, an API to describe hierarchical scientific data

The Mesh Blueprint, usage conventions, examples

14:00 Ascent, an in-situ visualization and analysis library using Conduit

Making images

Transforming data, extracting data

Queries and Conditional triggers

14:30 Instrumentation of simulation codes with Ascent

JacobiPython: Use jupyter notebooks on Piz Daint
SPH-EXA: an open-source parallel particle simulation code
LULESH: an open-source parallel unstructured flow solver

15:00 Coffee break

Agenda A.C.

15:30 ParaView Catalyst v2, an in-situ visualization and analysis library using Conduit

The ParaView interactive application, introduction

The Paraview parallel server architecture

The Catalyst API

The ParaView Catalyst Blueprint

Python scripting, Data Extractors

16:00 Instrumentation of simulation codes with Catalyst

SPH-EXA: an open-source parallel particle simulation code
LULESH: an open-source parallel unstructured flow solver
Catalyst, connecting to a live simulation, steering

ParaView Cinema: An Image-Based Approach to Extreme-Scale Data Analysis

16:25 Ascent executing ParaView Python code

16:35 Future developments, alternative workflows (in-transit visualization)

16:45-17:00 Wrap-up, Q&A

Technical details

- course account “class1??” with a password
- ssh ela.cscs.ch + ssh daint.cscs.ch
- https://jupyter.cscs.ch
- Your account has been already bootstrapped with requirements for jupyter lab
  - see $HOME/.local/shared/jupyter/kernels
- Reservation for computer nodes “in-situ”
- Your account will expire at the end of the day
- My repository should not expire: https://github.com/jfavre/InSitu-Vis-Tutorial2022

Overview

- What is in-situ visualization, why do we need it? What solutions are available to implement it?
- See recent book "In Situ Visualization for Computational Science"
- Some examples from the previous decade (libSim, Catalyst)

- Replay some of the presentations from the most recent Ascent tutorial at SC21

- Detail the ParaView-Catalyst and Ascent solutions, with practical examples.

“Piz Daint” at CSCS, the Swiss flagship for national HPC Service

- Cray XC40/XC50
- 5704 hybrid nodes (Intel Xeon E5-2690 v3/NVIDIA Tesla P100)
- 1813 multi-core nodes (Intel Xeon E5-2695 v4)

Coming online in 2023, the ‘Alps’ system infrastructure

will replace CSCS’s Piz Daint and serve as a general-purpose system.

Link

The past

- For decades, the dominant paradigm has been *post-hoc* visualization
- Simulation codes iterate, and save data at regular time intervals.
  - Visualization and domain scientists can then read the data back from storage and interactively explore the data without time constraints
  - In particular at CSCS, see https://user.cscs.ch/computing/visualisation/

“Without I/O, no visualization is possible”

The true cost of doing I/O is an aggregate of the solver’s I/O phase and the many iterations of visualization sessions.

Post-hoc visualization

Even if scientists could afford to keep most of the data for analysis, they must transfer the data to a machine with sufficient capacity and processing power:

➔ Very high data transfer
➔ Visualization machine needs to be almost as powerful as the supercomputer
➔ The alternative: use smaller temporal and spatial subsets

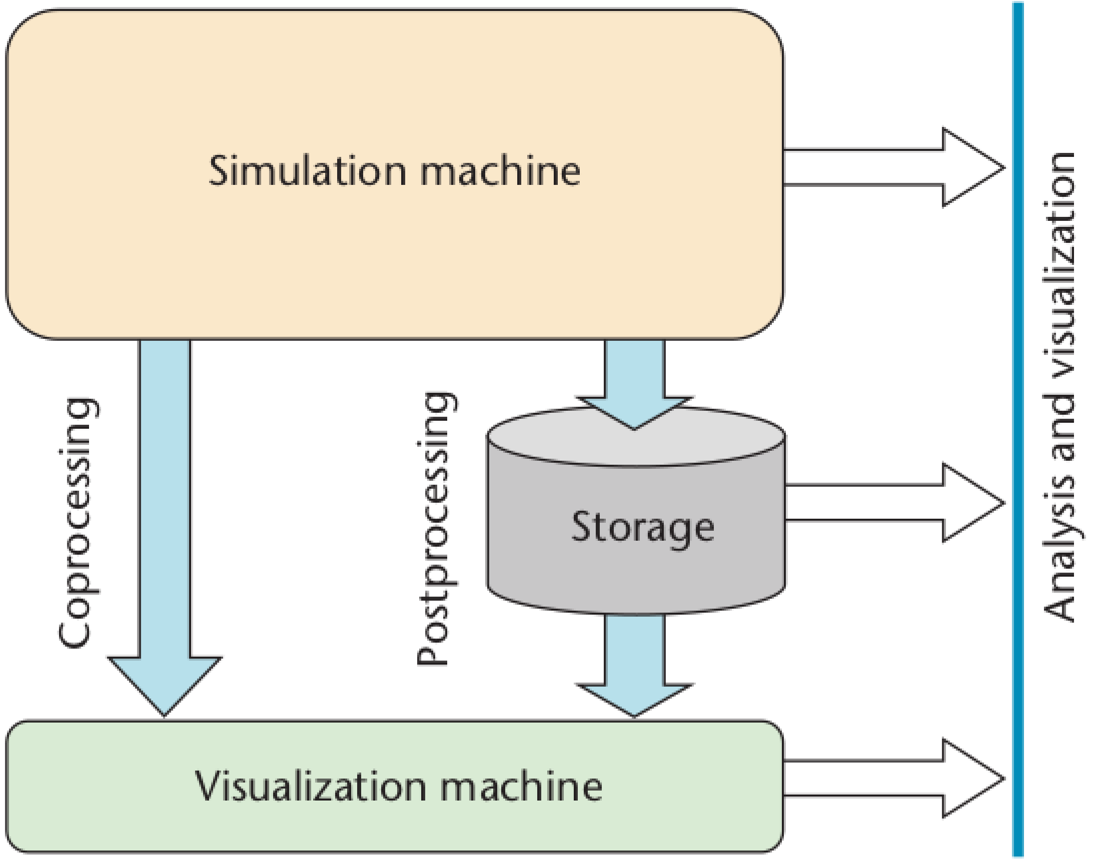


Figure taken from "*In Situ Visualization at Extreme Scale: Challenges and Opportunities*", Kwan-Liu Ma, IEEE CG&A, nov/dec 2009

in situ visualization

Instrument the code such that both the simulation and visualization calculations run on the same hardware

This runtime co-processing can render images directly or extract features -- *which are much smaller than the original raw data*

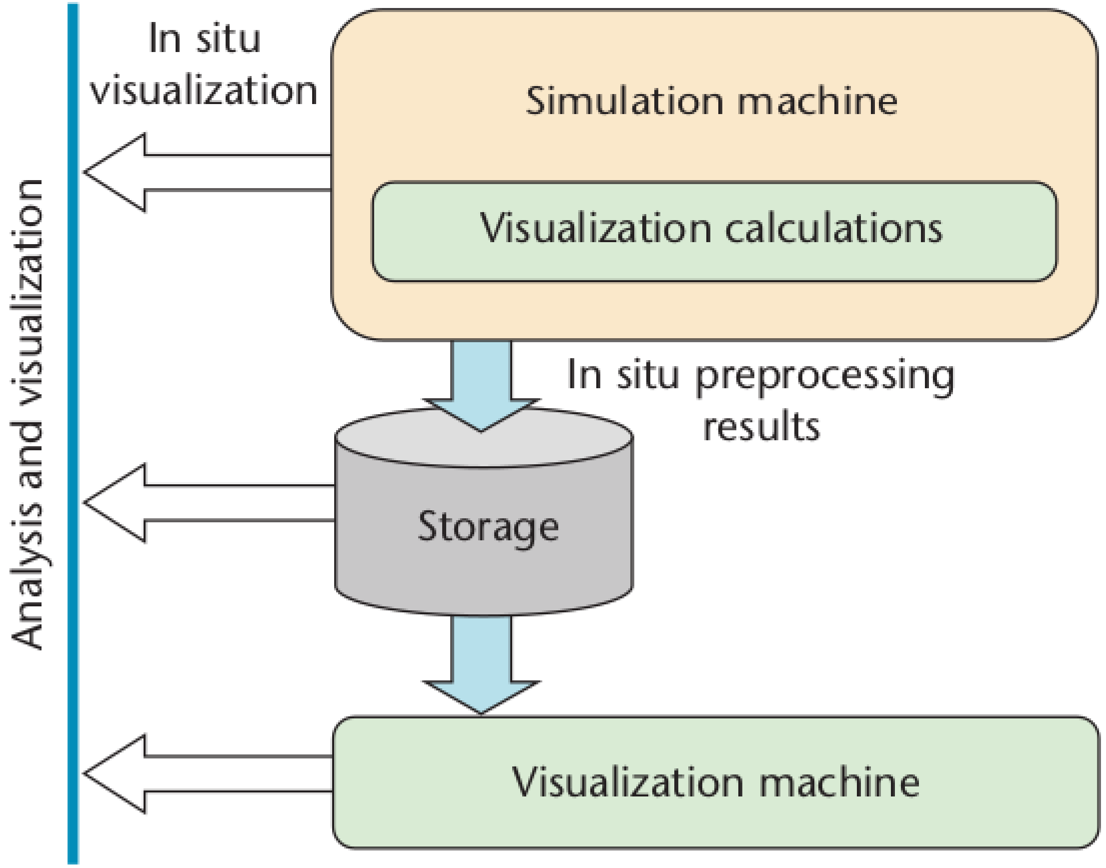


Figure taken from "*In Situ Visualization at Extreme Scale: Challenges and Opportunities*", Kwan-Liu Ma, IEEE CG&A, nov/dec 2009

In-situ visualization has raised quite a few questions

- Sharing physical resources and domain decomposition?
- What % of time can we afford to “do visualization” vs. “advance the solver”?
- Which feature extraction and visualization tasks are best suited for on-the-fly processing?
- Since less data would be effectively stored to disk, should we augment it with ancillary data?
- Can we provide a generic abstraction to describe the data and mesh structures?

A third paradigm also emerged: in-transit visualization

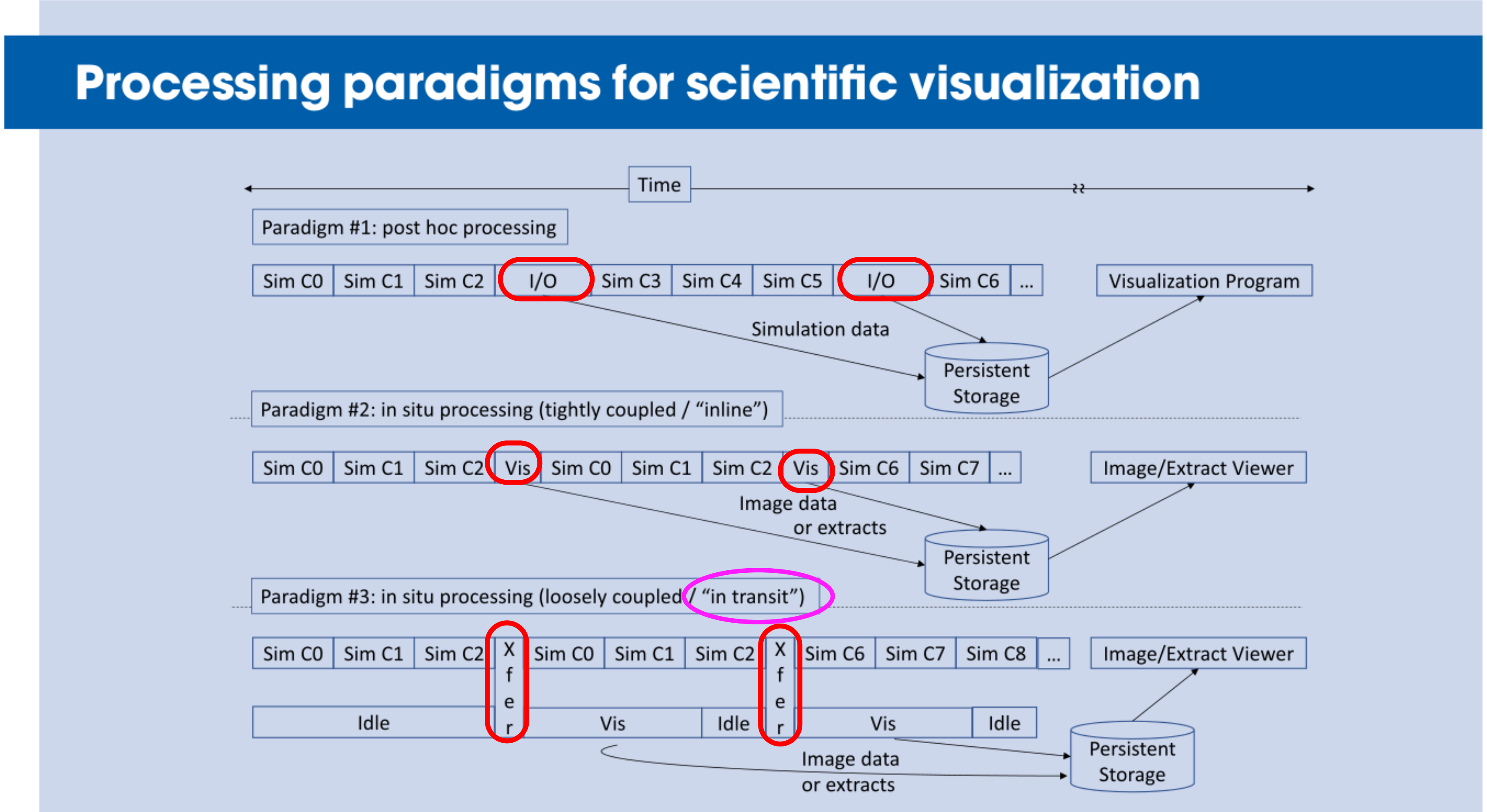


Figure taken from “*In Situ Visualization for Computational Science*”, Hank Childs et al., IEEE CG&A, nov/dec 2019

Does “in-situ” mean “in place”?

- The data is already in the processor’s memory space, without touching the disks

- If simulation data is moved to a distinct set of resources (nodes dedicated to visualization), we are still analysing data “in place”, but is it “in-situ”?

- There has been quite a few variants on the terminology:
  - Co-processing, concurrent processing, run-time visualization

Many definitions and colloquial use for “in-situ”

- An exhaustive panorama of the different systems in use was created :
- "A Terminology for In Situ Visualization and Analysis Systems“, Hank Childs et al, International Journal of High Performance Computing Applications, 34(6):676–691
- http://cdux.cs.uoregon.edu/pubs/ChildsIJHPCA.pdf

- For the scope of this paper, “in situ processing” was defined to be:

“processing data as it is generated”

in situ systems were best described via multiple, distinct axes

- integration type

  How visualization and analysis code is integrated with the simulation code?
- proximity

  How close is the visualization code from the data?
- Access

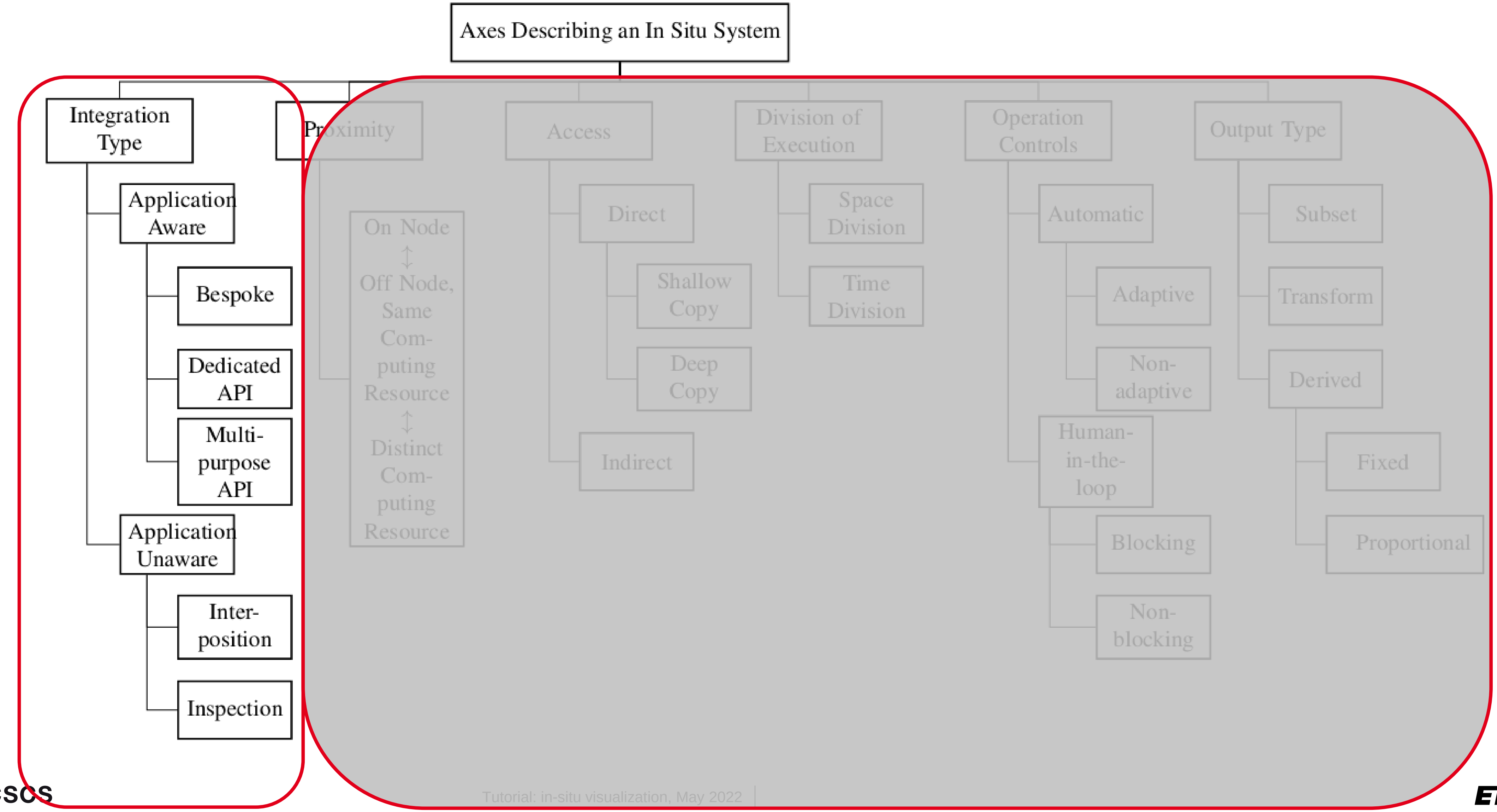
  How does the simulation give access to the data?
- division of execution:

  how compute resources are shared between simulation and in situ routines.
- operation controls:

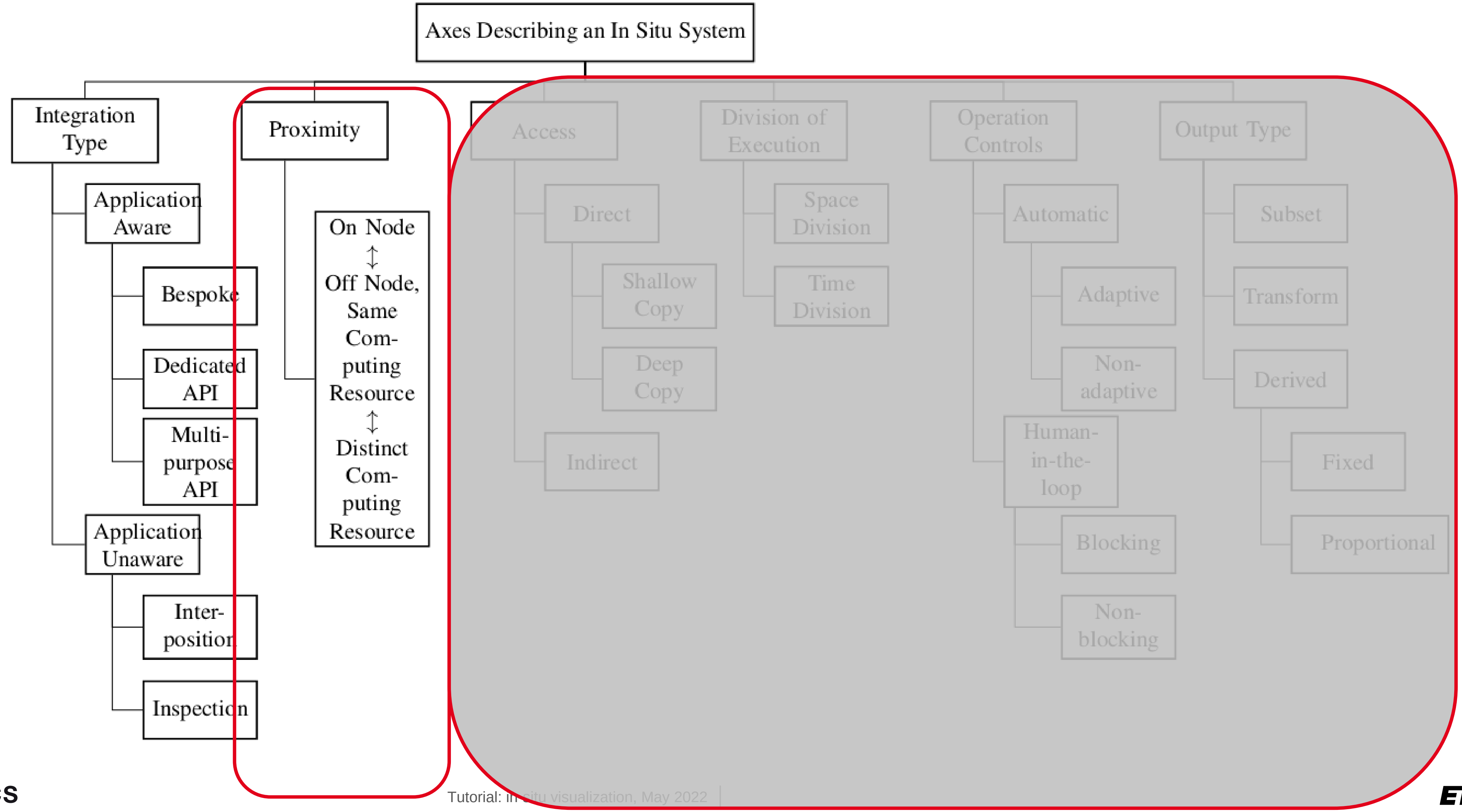
  the mechanism for selecting which operations are executed during run-time
- output type

  which types of operations are performed on the simulation data before it is output.

Integration Type

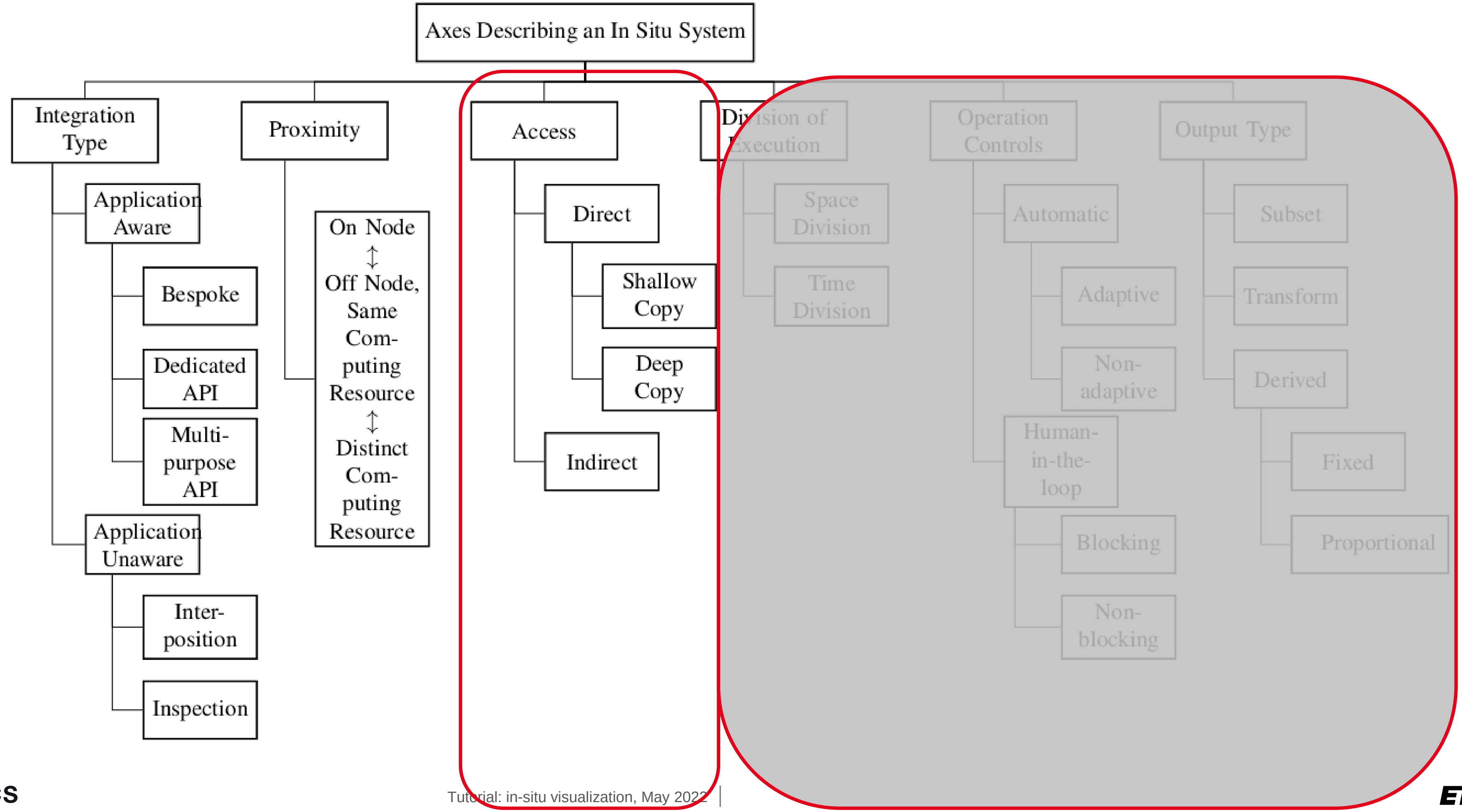


Proximity



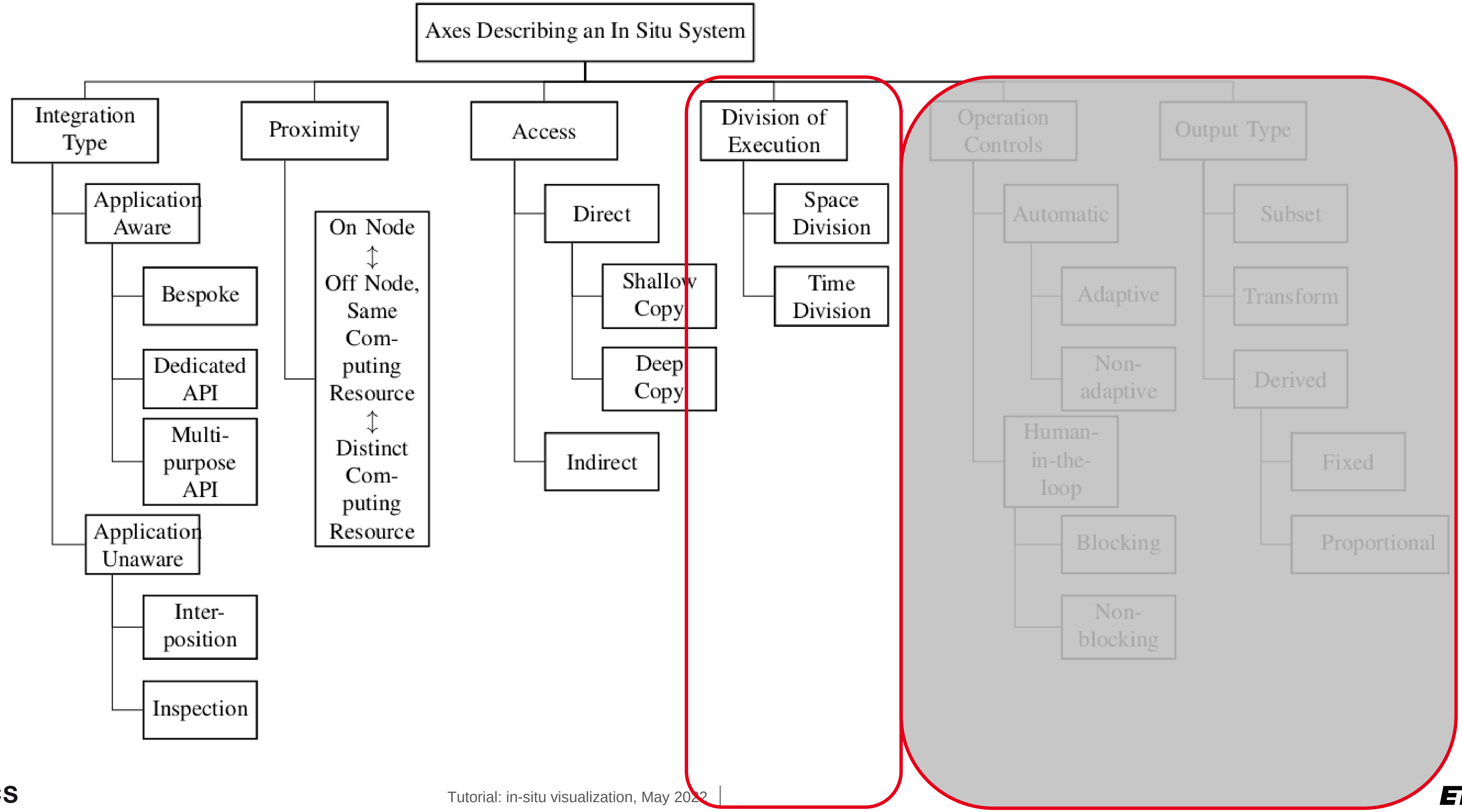
Axes Describing an In Situ System
Integration Type
Application Aware
Bespoke
Dedicated API
Multi-purpose API
Application Unaware
Inter-position
Inspection
Proximity
On Node ↕ Off Node, Same Computing Resource ↕ Distinct Computing Resource
Access
Direct
Shallow Copy
Deep Copy
Indirect
Division of Execution
Space Division
Time Division
Operation Controls
Automatic
Adaptive
Non-adaptive
Human-in-the-loop
Blocking
Non-blocking
Output Type
Subset
Transform
Derived
Fixed
Proportional

Access

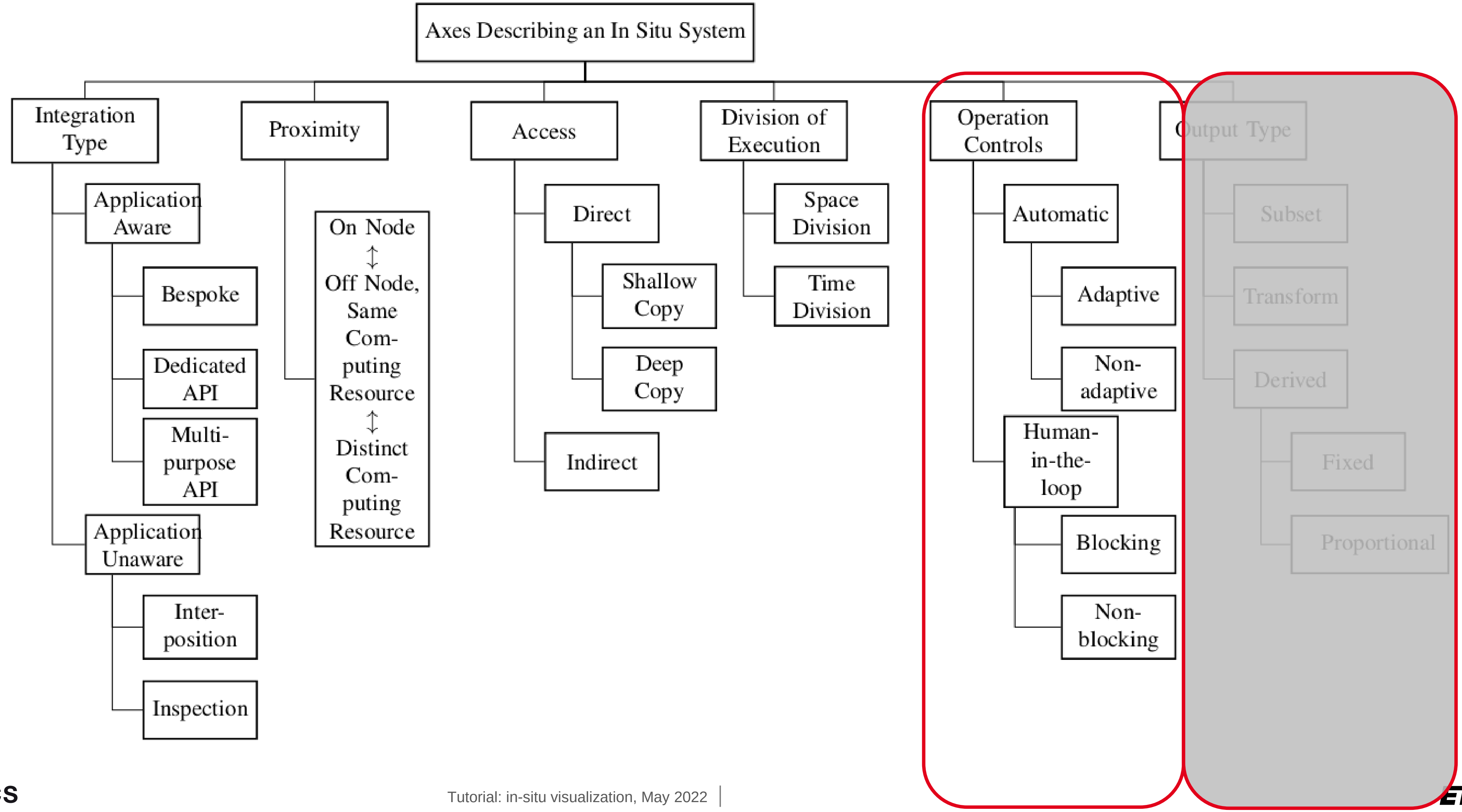


Axes Describing an In Situ System
Integration Type
Application Aware
Bespoke
Dedicated API
Multi-purpose API
Application Unaware
Inter-position
Inspection
Proximity
On Node ↕ Off Node, Same Computing Resource ↕ Distinct Computing Resource
Access
Direct
Shallow Copy
Deep Copy
Indirect
Division of Execution
Space Division
Time Division
Operation Controls
Automatic
Adaptive
Non-adaptive
Human-in-the-loop
Blocking
Non-blocking
Output Type
Subset
Transform
Derived
Fixed
Proportional

Division of Execution



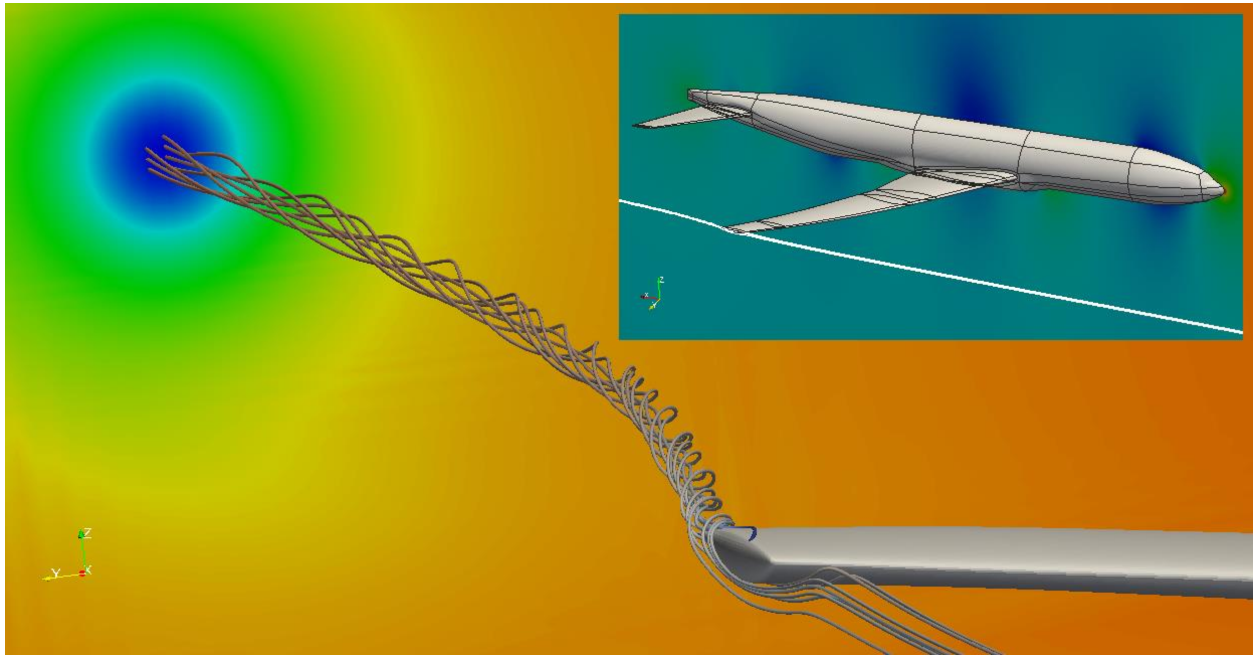
Operation Controls



Feature extraction?

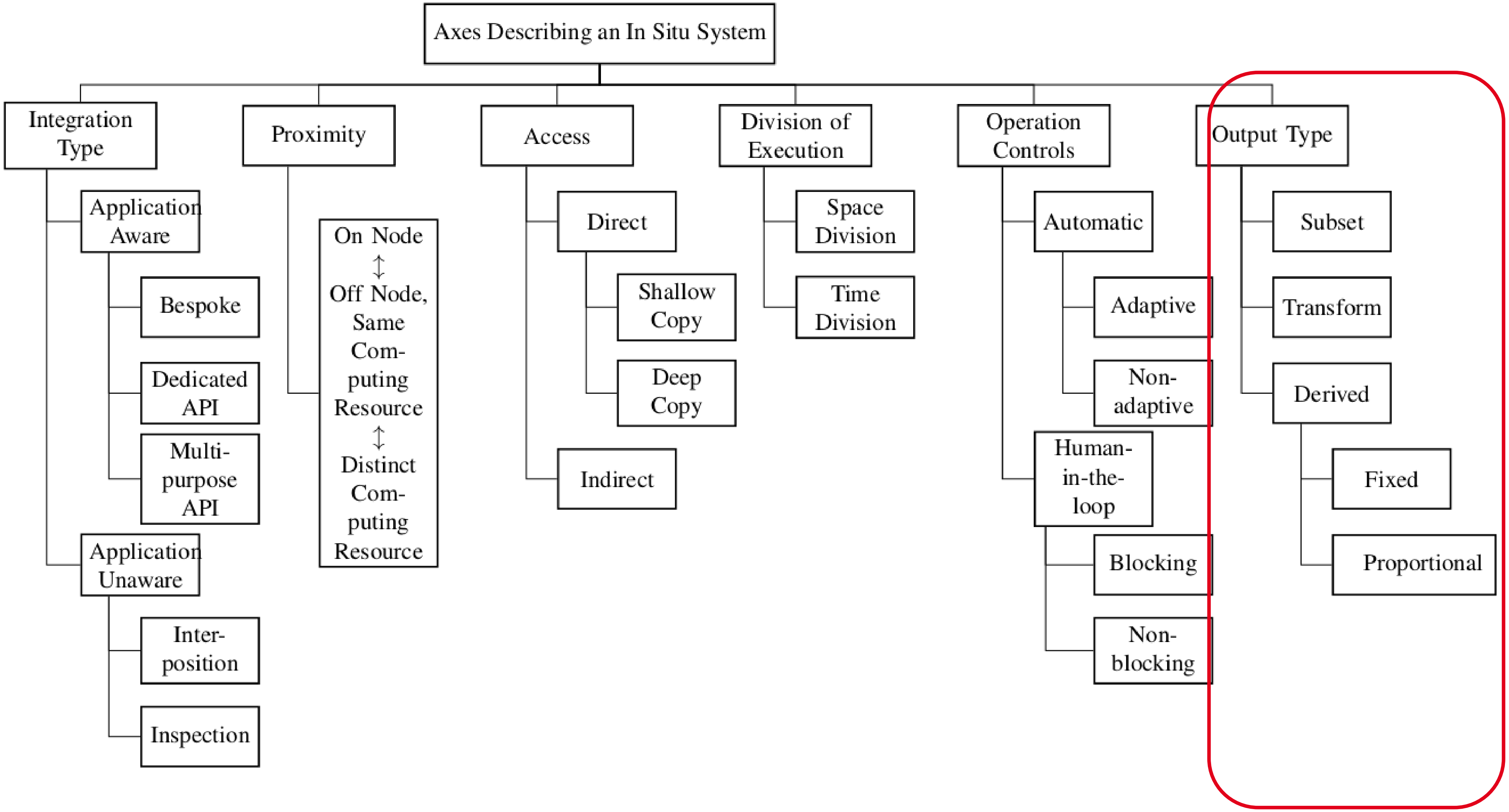
- What's a feature?

- Some small[er] data worth of interest because of domain knowledge



- Without "interactive" data exploration, it can be difficult to know a-priori which features to extract, or how to tune parameters.
- Knowing when to trigger a potentially expensive calculation would also be handy.

Output types



Outcome

- In that paper, 15 existing systems were reviewed. Half of them implement a Time Division, with On Node proximity.
  - The simulation advances, then pauses and hands control to the in situ system which completes its operations, hands control back, and so on
  - For example: Ascent, ParaView-Catalyst, Visit/Libsim
- Other examples, ADIOS, SENSEI, which can operate in different modes, sending also their data to distinct vis resources
- ADIOS, for example, can integrate with other workflow or data analytics systems without detailed knowledge of the underlying software and hardware stack
- ADIOS allows users to combine data storage, data staging, data compression, and/or data reduction (ZFP, SZ, BZip2 (compression), FlexPath, Dataspace (data staging), and coupling with ParaView, Visit, Ascent)

SENSEI

SENSEI provides a generic API to enable a “write-once, run-anywhere” environment. This approach focuses on data proximity and portability, runtime selection between running On Node or Off Node

SENSEI provides access to a diverse set of in situ analysis back-ends and transport layers such as ADIOS, Libsim, Ascent, Catalyst etc , through a simple API and data model.

Simulations instrumented with the SENSEI API can process data using any of these back-ends interchangeably. The back-ends are selected and configured at run-time via an XML configuration file.

For more details, refer to the SC2021 tutorial notes

What about my [the solver] internal data model?

- Do the standard visualization applications support all the data structures I use in my code?
  - => Use Data Adaptors
- Are our standard visualization applications ready to handle new alternate data representations?
  - => Use Data Converters

- In recent years, we have seen new ways of thinking about data simulations (run multiphysics code, run ensembles, use ML).
- => new data, perhaps quite different from the traditional "mesh of gridded points"

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Conduit: introduction

Conduit: Simplified Data Exchange for HPC Simulations

- Conduit is an open source project from Lawrence Livermore National Laboratory that provides an intuitive model for describing hierarchical scientific data in C++, C, Fortran, and Python. It is used for data coupling between packages in-core, serialization, and I/O tasks.

- Conduit provides a convention to describe computational simulation meshes. This is called the Mesh Blueprint.
- Illustration of Mesh Blueprint examples

- Ascent and Catalyst use Conduit for describing data and other parameters which can be communicated between a simulation and the visualization apps.

Conduit by examples

Setup from a terminal

ssh class1??@daint.cscs.ch

git clone https://github.com/jfavre/InSitu-Vis-Tutorial2022

Read and execute contents from “PizDaint_Instructions.txt”

Conduit by examples

We will use jupyter lab and our course accounts to get compute nodes on Piz Daint

Username: class1??

Password: ???????

Reservation: in_situ

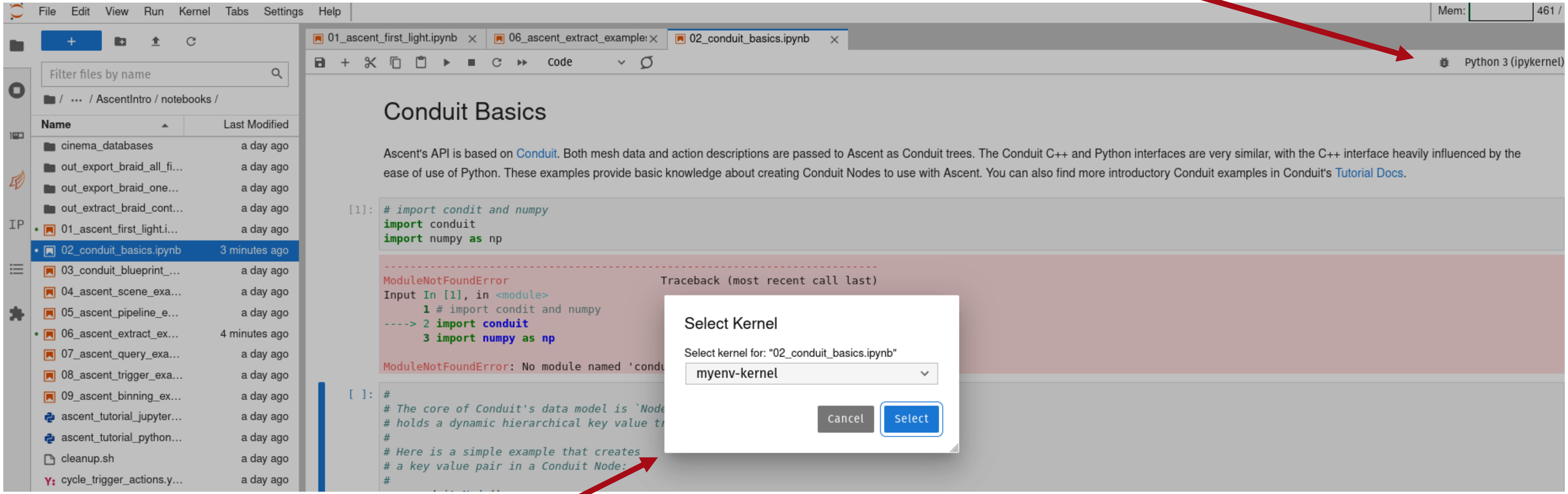
See https://user.cscs.ch/tools/interactive/jupyterlab/ and get access at

https://jupyter.cscs.ch

goto AscentIntro/notebooks

execute 02_conduit_basics.ipynb

import Error?



Change kernel to *myenv-kernel*

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Conduit + Ascent

Ascent

Ascent is an easy-to-use flyweight in situ visualization and analysis library for HPC simulations:

- Supports: Making Pictures, Transforming Data, and Capturing Data for use outside of Ascent
- Young effort, yet already includes most common visualization operations
- Provides a simple infrastructure to integrate custom analysis
- Provides C++, C, Python, and Fortran APIs
- Ref

"*The ALPINE in situ infrastructure: Ascending from the ashes of strawman*", M. Larsen et al., Proc. 3rd Workshop In Situ Infrastructures Enabling Extreme Scale Anal. Vis. Denver, CO, USA, Nov. 12–17, 2017

Ascent

Ascent slides from the recent SC2021 tutorial

Thanks go to Cyrus Harrison and colleagues.

Ascent

Ascent is based on several components:

- The Conduit Mesh Blueprint!
- Runtimes providing analysis, rendering and I/O
  - Runtimes will execute a number of *actions*, defined by Conduit Nodes
- Data Adaptors (internal)

Ascent tutorial pages (Thanks to LLNL)

- Tutorial link
- SC2021 tutorial

- docker run -p 8888:8888 -t -i alpinedav/ascent-jupyter

Ascent tutorial first example

goto AscentIntro/notebooks

Run 01_ascent_first_light.ipynb

Ascent tutorial examples,

Generating time dependent data

- Run 03_conduit_blueprint_mesh_examples.ipynb

Rendering images with Scenes

- Run 04_ascent_scene_examples.ipynb

Transforming data with Pipelines

- Run 05_ascent_pipeline_examples.ipynb

Running with different visualization pipelines

In its basic form, we usually have a default scene description into the Ascent bridge code

- At run-time, we can supply different scenes, with additional pipelines
- If present, a file called "ascent_actions.{json,yaml}" will be read, overriding the actions previously set

- http://www.yamllint.com/ is your friend

- See example in Examples/LULESH/{ascent_actions.yaml,trigger_ascent_actions.yaml}

Let’s add a scene description

```
"action": "add_scenes",
        "scenes": {
           "s1": {
              "plots": {
                 "p1": { "type": "pseudocolor", "field":
"Density"} },
              "renders": {
                 "r1": {
                   "image_prefix": “DensityImage.%05d",
                   "camera": {
                     "look_at": [0, 0, 0],
                     "position": [-2.17, 1.79, 1.80],
                     "up": [0.44, 0.84, -0.30]
                   }
```

```
-
action: "add_scenes"
scenes:
  s1:
    plots:
      p1:
        type: "pseudocolor"
        field: "Density"
   renders:
      r1:
        image_prefix: “DensityImage.%05d"
        camera:
          azimuth: 30
          elevation: 11
```

Let’s add a pipeline description

```
"action": "add_pipelines",
        "pipelines": {
          "pl1": {
            "f1": {
              "type": "threshold",
              "params": {
                "field": "Density",
                "min_value": 1.4,
                "max_value": 2000
               }
            }
          }
        }
},
```

The scene description is refined with the new pipeline

```
"action": "add_scenes",
        "scenes": {
           "s1": {
              "plots": {
                 "p1": {
                    "type": "pseudocolor",
                    "pipeline": "pl1",
                    "field": "Density"
                  }
              },
```

Example: Instrument an SPH simulation package with Ascent

- The smooth particle hydrodynamics (SPH) technique is a purely Lagrangian method. SPH discretizes a fluid in a series of interpolation points whose distribution follows the mass density of the fluid.

- PASC, the Swiss Platform for Advanced Scientific Computing initiative, supports the SPH-EXA project developing an SPH library.
- SPH-EXA is a C++17 headers-only code with no external software dependencies. The parallelism is currently expressed via the following models: MPI, OpenMP, CUDA and HIP.

Instrument the SPH-EXA simulation package with Ascent

- Define a Conduit mesh definition
- Define a Conduit scene definition

About 150 lines of code. Total!

Using Conduit, a particle set is trivially described [the coordinates]

```
particle_set = """
 coordsets:
   coords:
     type: "explicit"
     values:
     x: [0.0, 10.0, 20.0, 30.0]
     y: [0.0, 10.0, 20.0, 30.0]
     z: [0.0, 10.0, 20.0, 30.0]
"""
```

```
conduit::Node mesh;
mesh["state/cycle"].set_external(&d.iteration);
mesh["state/time"].set_external(&d.ttot);
mesh["coordsets/coords/type"] = "explicit";
mesh["coordsets/coords/values/x"].set_external(&d.x);
mesh["coordsets/coords/values/y"].set_external(&d.y);
mesh["coordsets/coords/values/z"].set_external(&d.z);
// The heavy-data is available via shallow-copy links
```

Using Conduit, a particle set is trivially described [the topology]

```
particle_set = """
  topologies:
    mesh:
      type: "unstructured"
      elements:
      shape: "point"
      connectivity: [0, 1, 2, 3]
      coordset: "coords"
"""
```

```
mesh["topologies/mesh/type"].set("unstructured");
mesh["topologies/mesh/elements/shape"].set("point");
mesh["topologies/mesh/coordset"].set("coords");

std::vector<int> conn(N); // N is # of particles
std::iota(conn.begin(), conn.end(), 0);
mesh["topologies/mesh/elements/connectivity"].set(conn);
```

Using Conduit, a particle set is trivially described [the solution fields]

```
particle_set = """
fields:
  rho:
    association: "vertex"
    values: [-1, -2, -3, -4]
    topology: "mesh"
    volume_dependent: "false"
    units: "g/cc"
"""
```

```
auto fields = mesh["fields"];
// Density scalar field
fields["rho/association"].set("vertex");
fields["rho/topology"].set("mesh");
fields["rho/volume_dependent"].set("false");
// Conduit supports shallow copy
fields["rho/values"].set_external(&d.rho);
```

Use one of conduit's relay protocol to save data to disk?

c++: extract node from SPH-EXA

In Python

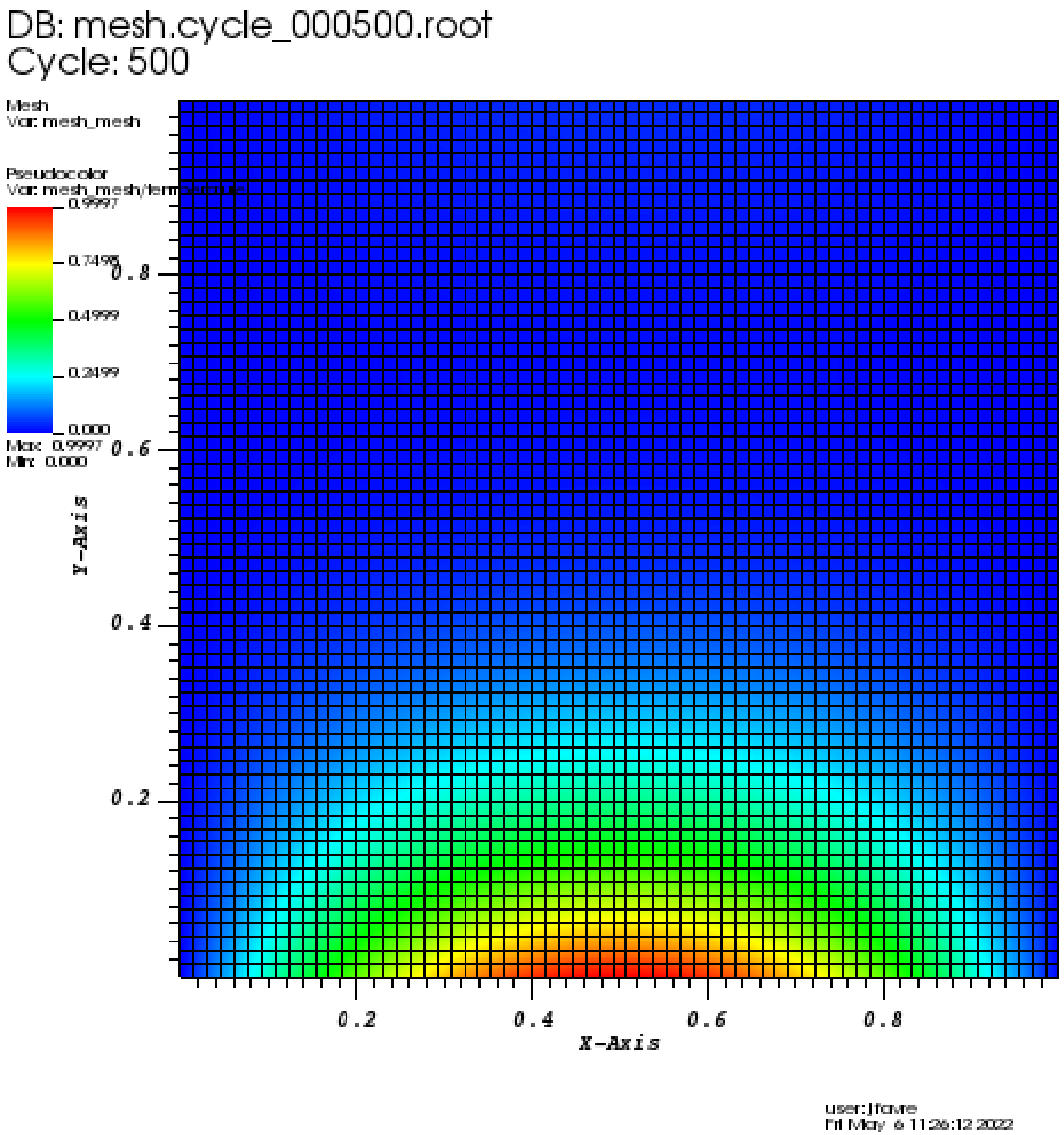
```
import conduit.relay as relay
  if relay.io.about()["protocols/hdf5"] == "enabled":
  relay.io.save(mesh, "/dev/shm/foo.hdf5")
```

```
h5ls -r /dev/shm/foo.hdf5
/                        Group
/coordsets               Group
/coordsets/coords        Group
/coordsets/coords/type   Dataset {9}
/coordsets/coords/values Group
/coordsets/coords/values/x Dataset {4}
…
/fields                  Group
/fields/rho              Group
/fields/rho/values       Dataset {4}
….
/topologies              Group
/topologies/mesh         Group
/topologies/mesh/coordset Dataset {7}
/topologies/mesh/elements Group
/topologies/mesh/elements/connectivity Dataset {4}
```

Supplementary material

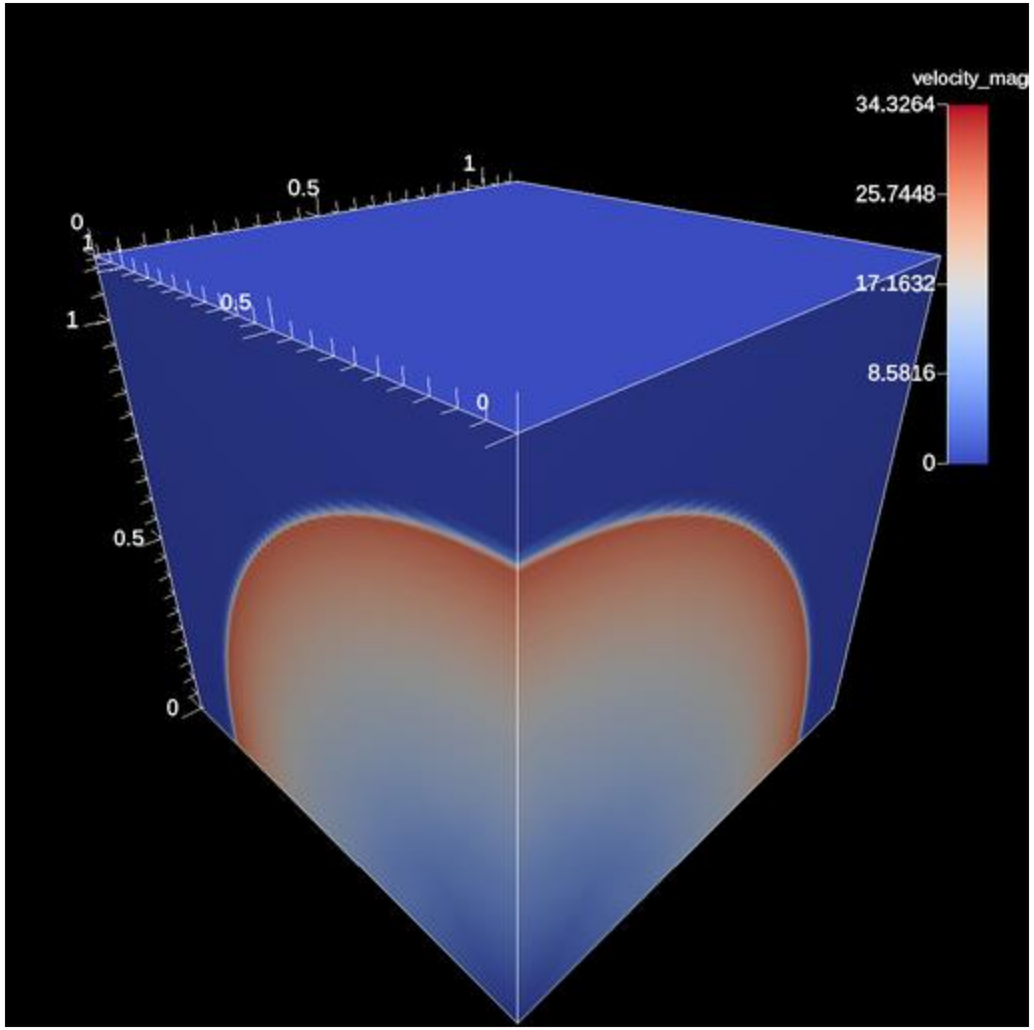
Instrument the Jacobi-Python example with Ascent

- Define a Conduit mesh definition
- Define a Conduit scene definition



Instrument the LULESH example with Ascent

- Define a Conduit mesh definition
- Define a Conduit scene definition



Coffee Break

Before jumping to ParaView Catalyst, let’s first introduce ParaView

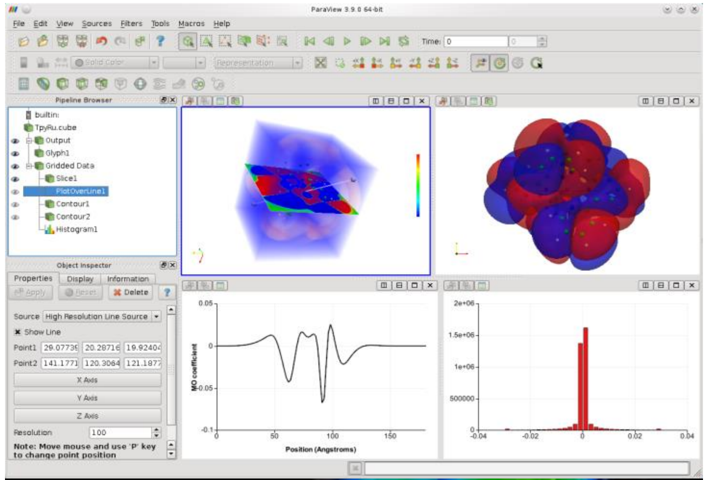
ISC 22 - Catalyst 2

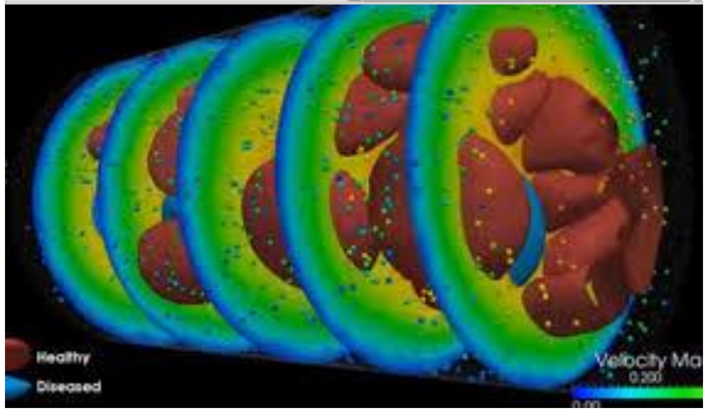
François Mazen

ParaView

ParaView / High-Performance Post-Processing

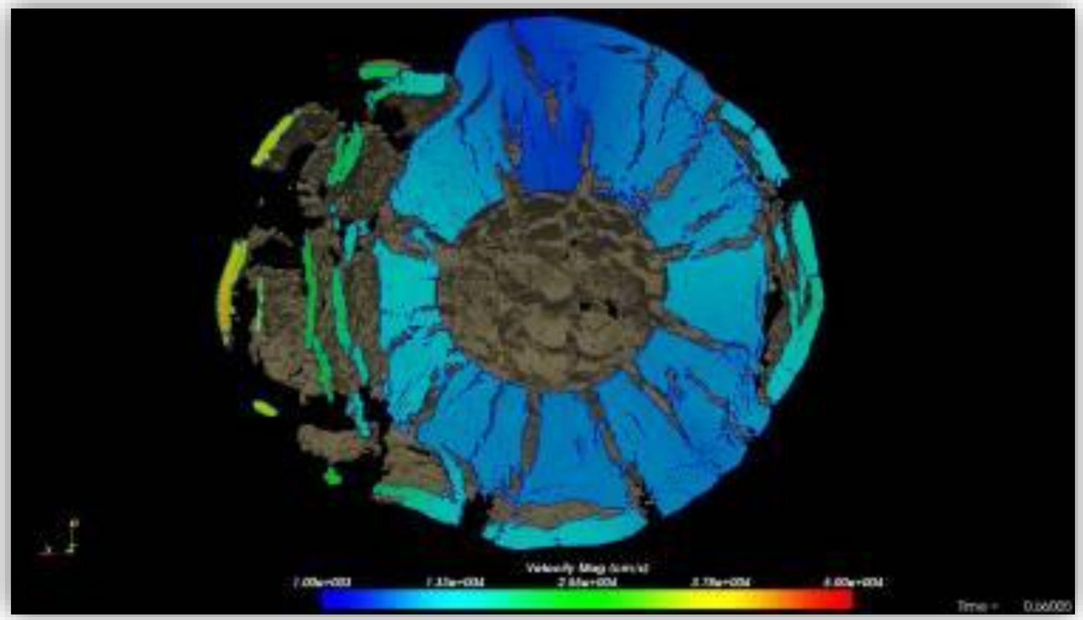
- **Open-source, multi-platform, data analysis and visualization application**
- **Analysis of extremely large datasets using distributed memory computing resources**
- Over 20 years of development
- Contributions from over 270 developers
- Over 1.6 million lines of code
- Over 150k yearly downloads
- www.paraview.org



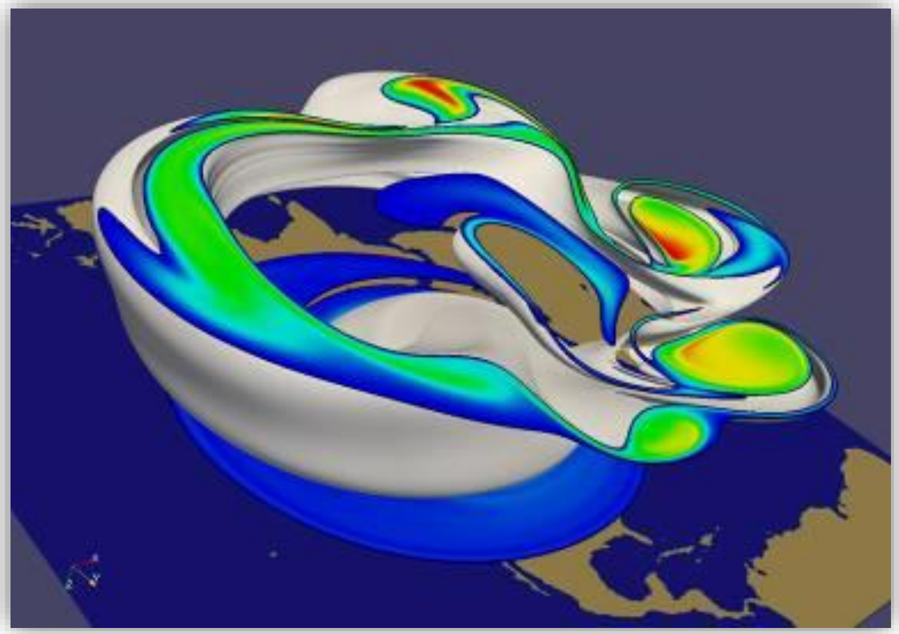


Extremely Large Data

1 billion cell asteroid detonation simulation



½ billion cell weather simulation



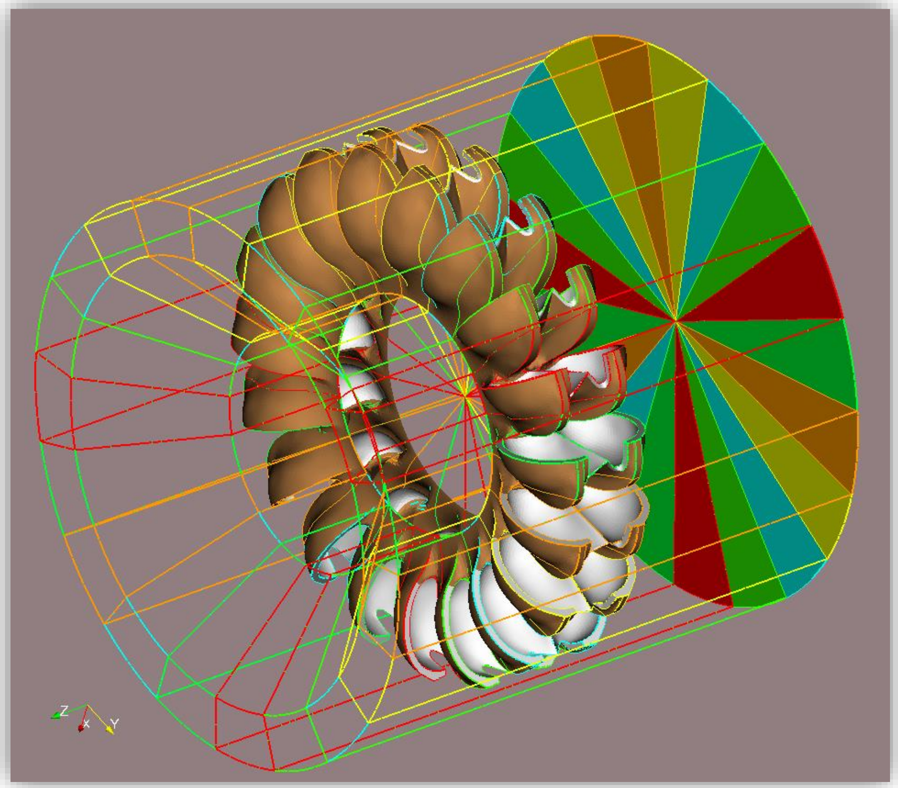
Source: Sandia National Lab

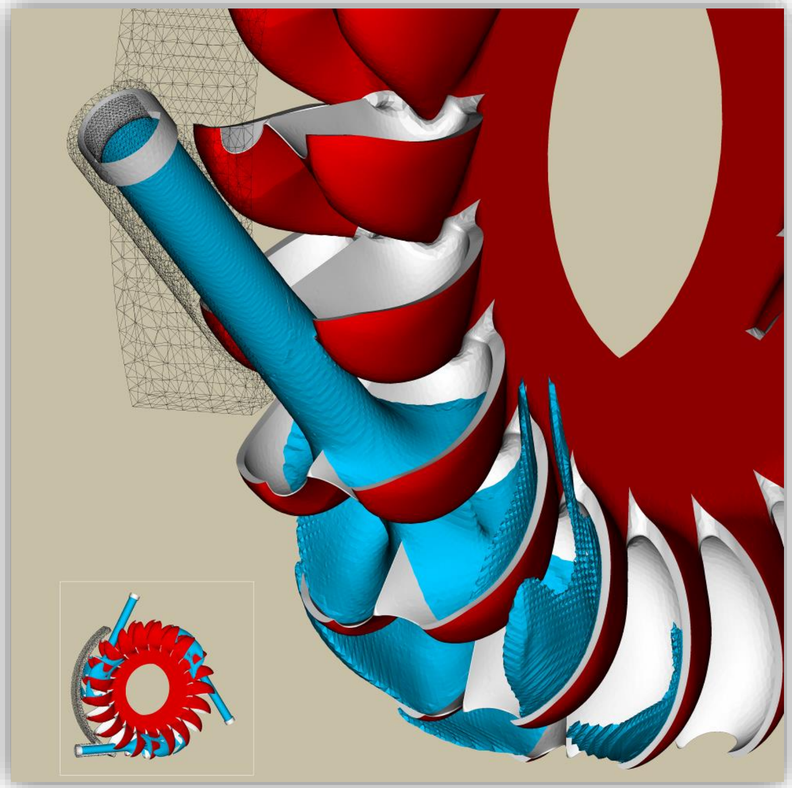
Sandia National Laboratories

kitware

Fast Large Data Interaction

CFD simulation of 20-30 million cells
with load balancing

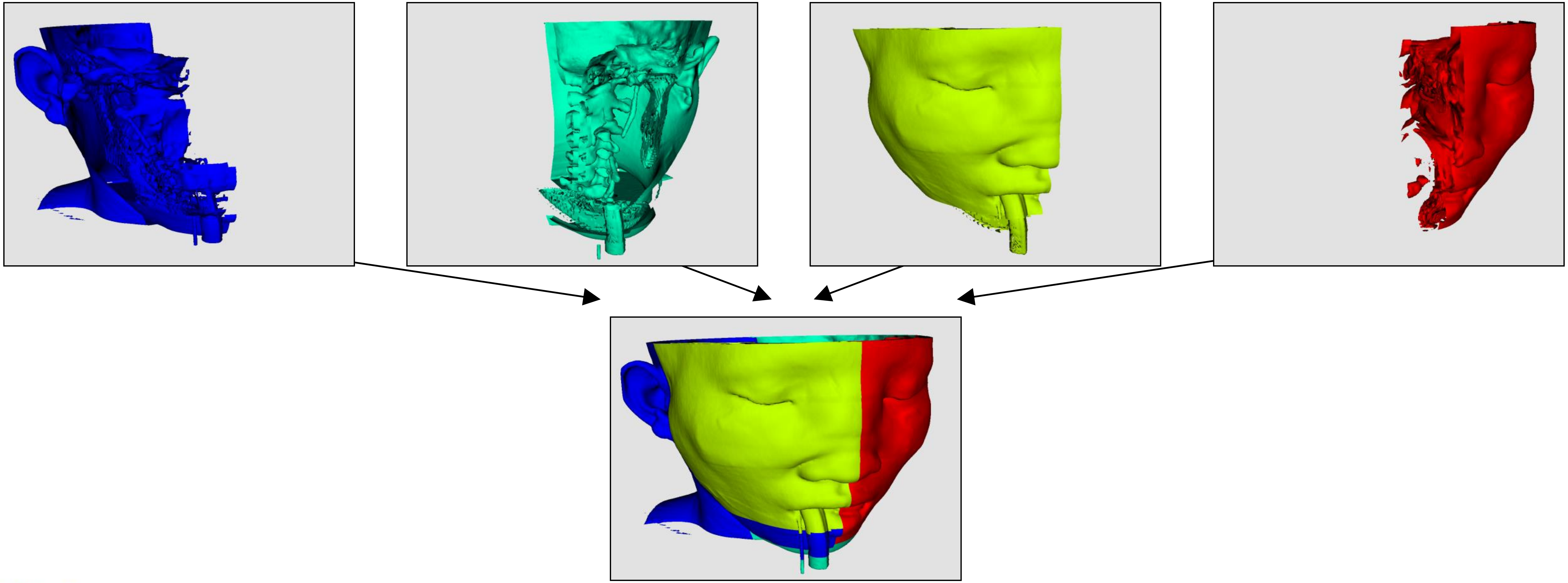




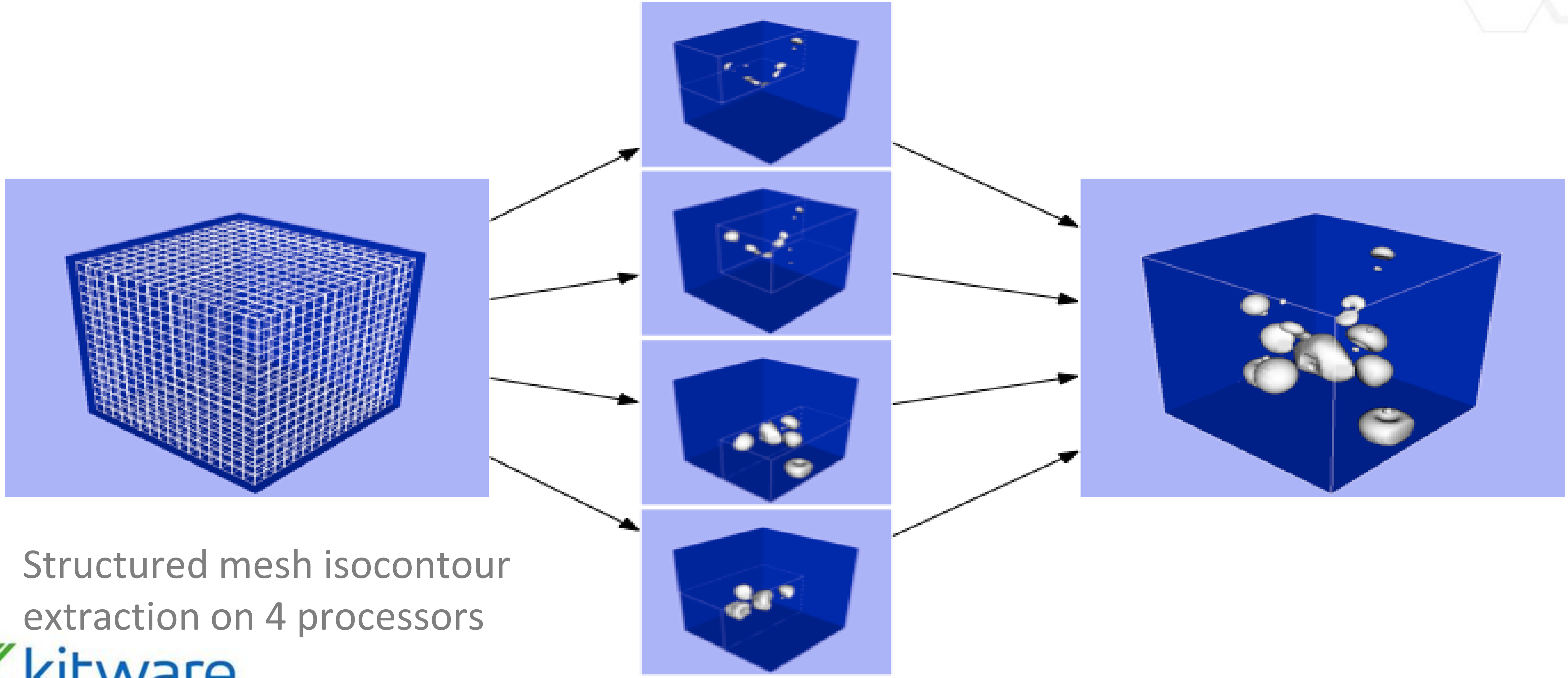
Source: Swiss National
Supercomputing center

Kitware

Parallel Rendering

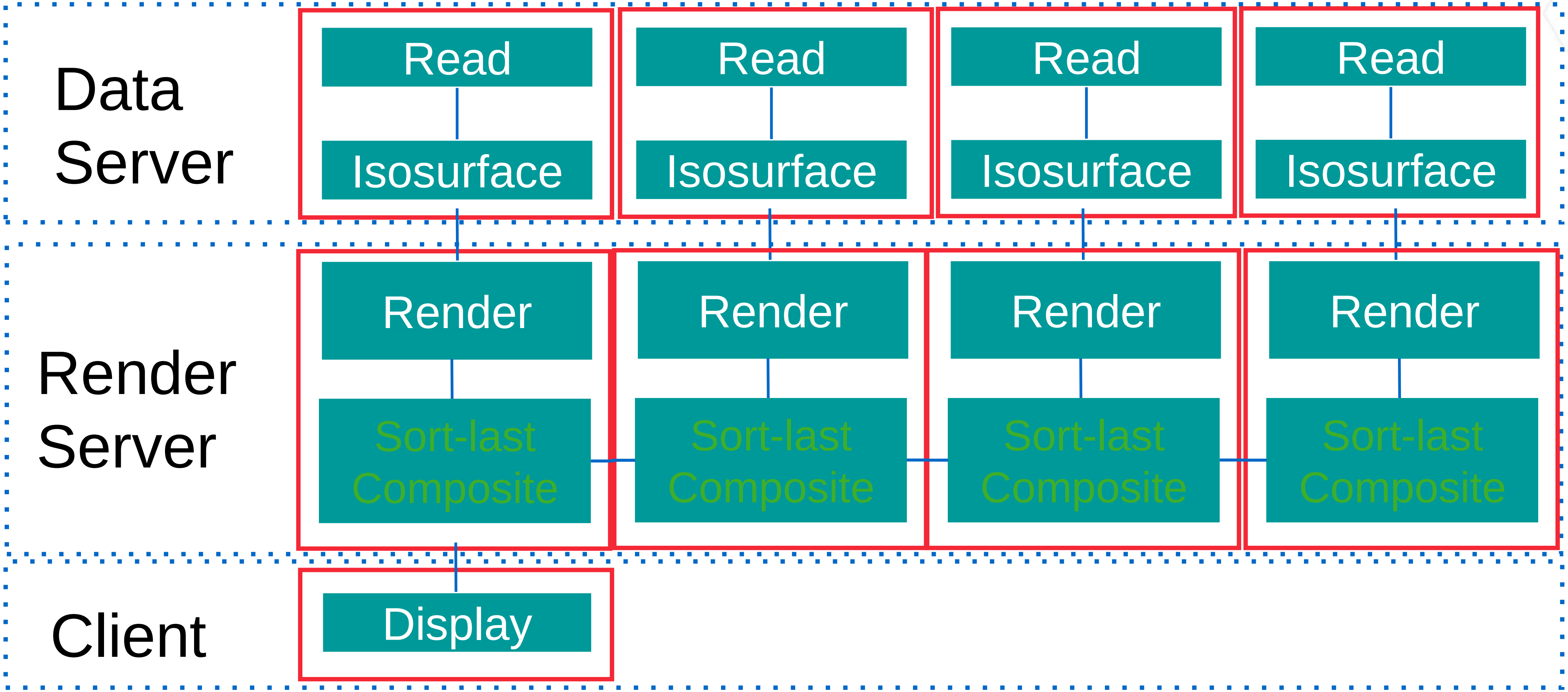


Distributed Processing



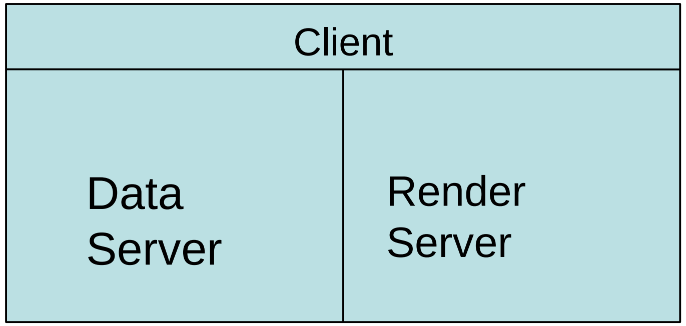
Structured mesh isocontour extraction on 4 processors

Data Parallel Pipelines

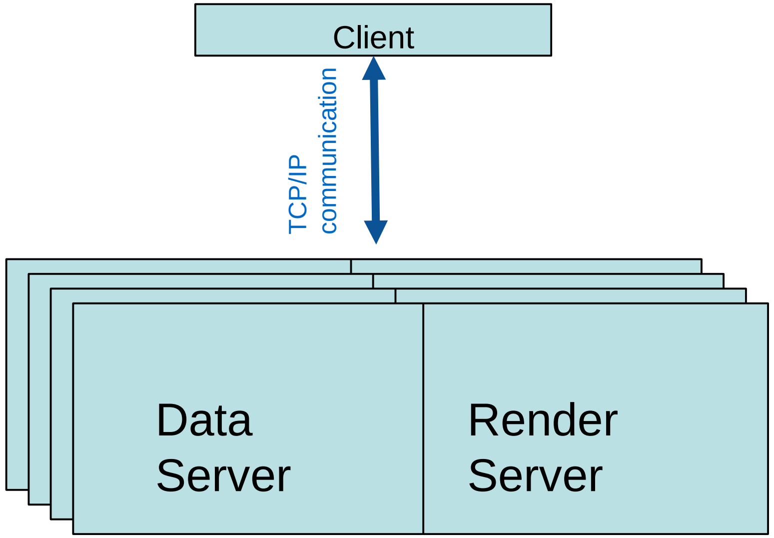


Deployment Topologies

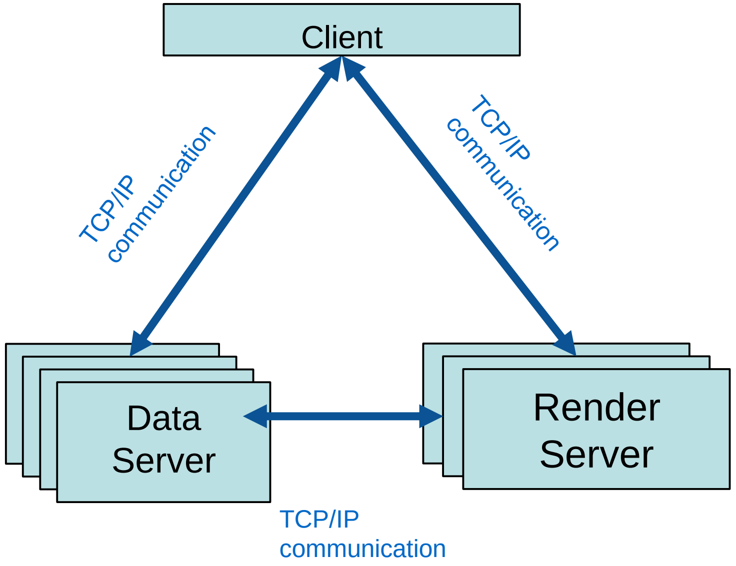
Builtin Server



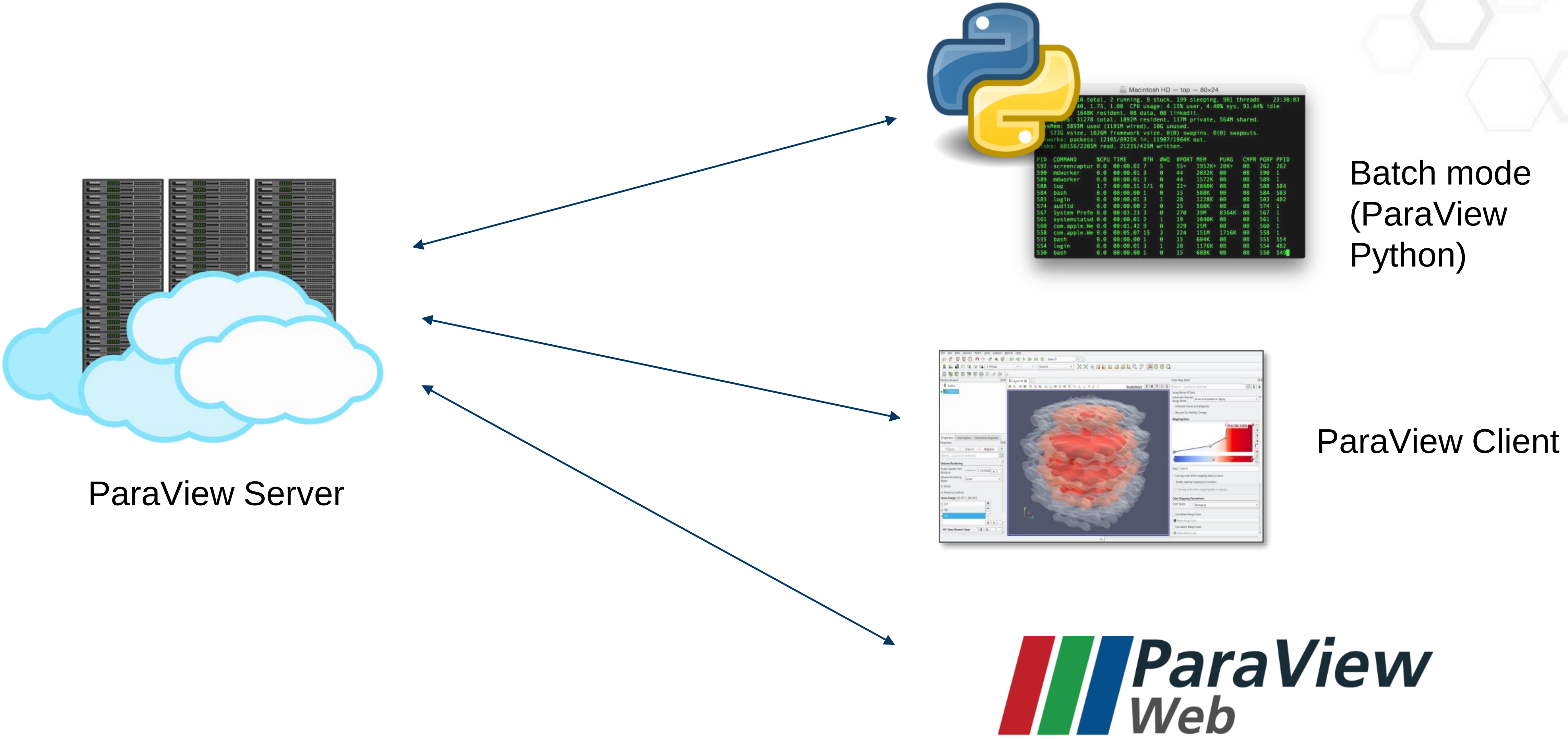
Client-Server



Client-Render Server-Data Server



Running ParaView

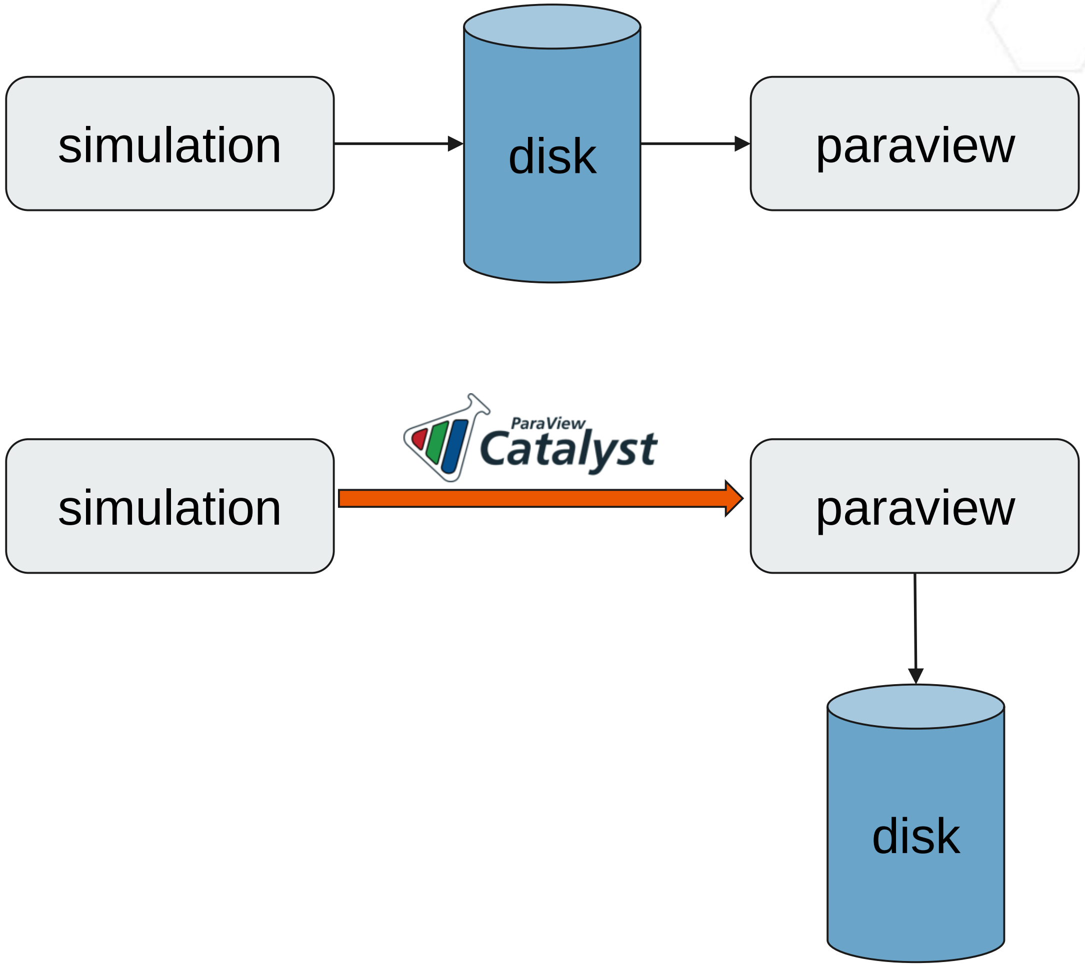


In-Situ Analysis

What is ParaView Catalyst?

- **An in-situ framework embedded with ParaView**
  - in-situ analysis: uses ParaView analysis capabilities
  - live visualization: uses ParaView to connect to the simulation
  - easy configuration: uses ParaView to generate scripts
  - open source

- **Challenges : explosion of scientific data**
  - Storage
  - Bandwidth

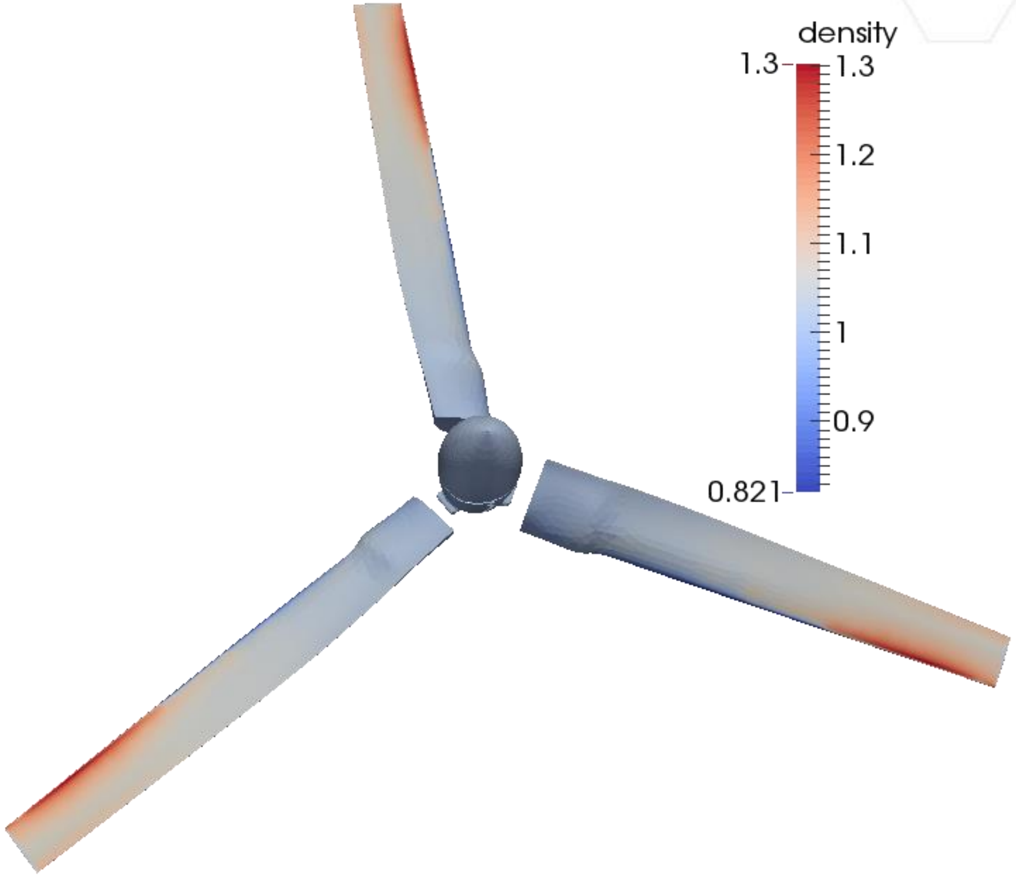


kitware

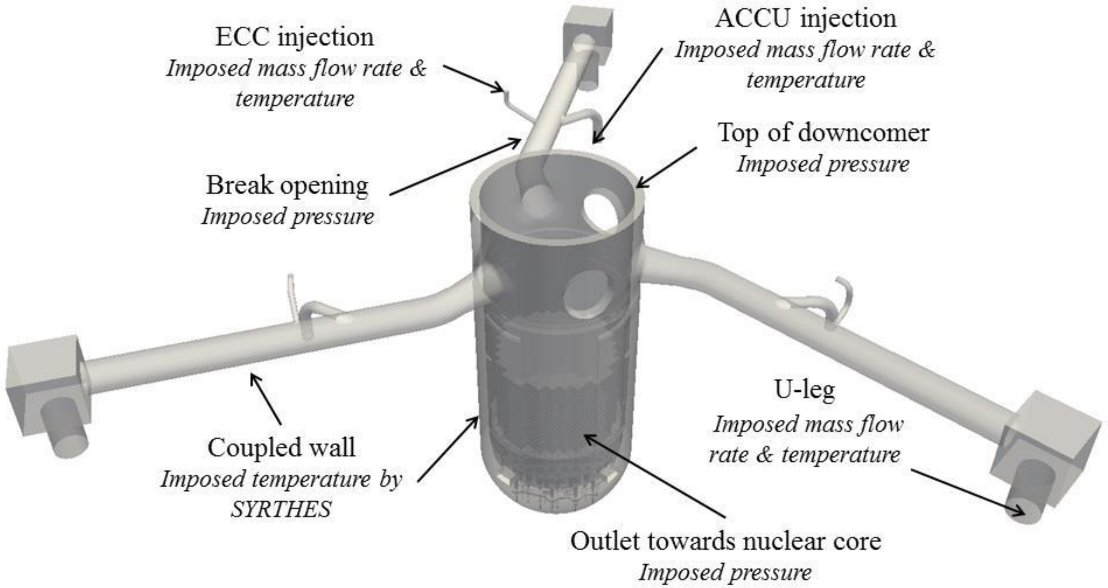
Saving Data Storage

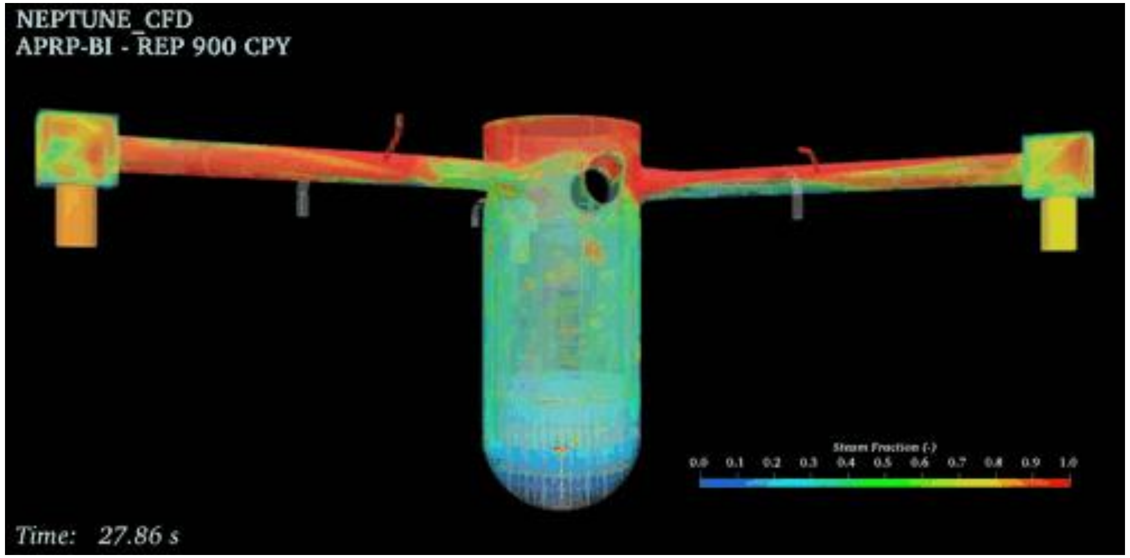
Rotorcraft simulation output size for a single time step:

- **Full data set – 448 MB**
- **Surface of blades – 2.8 MB**
- **Image – 71 KB**



ParaView Catalyst in Code_Saturne Example

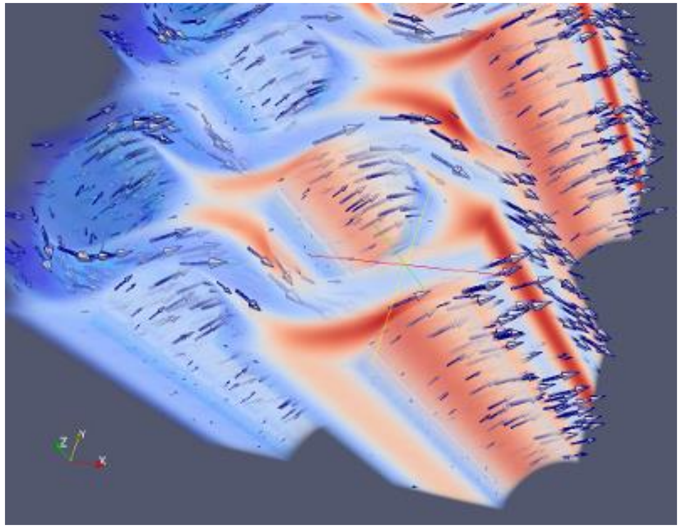




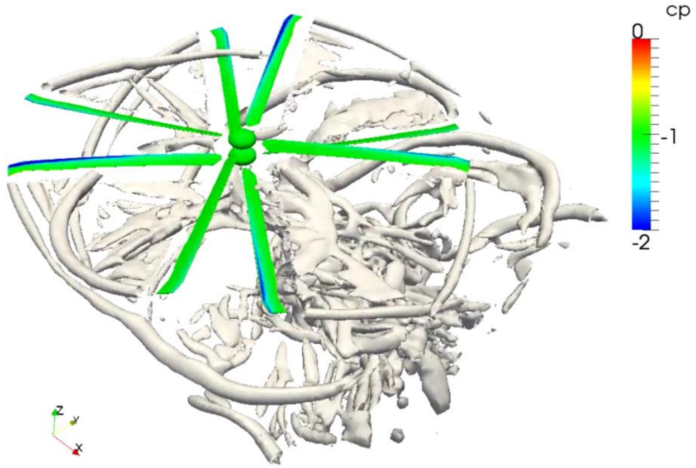
- **Fluid mesh: about 6 500 000 cells**
- **Solid mesh: around 4 000 000 cells**

Courtesy of Nicolas MERIGOUX & Yvan FOURNIER, EDF R&D

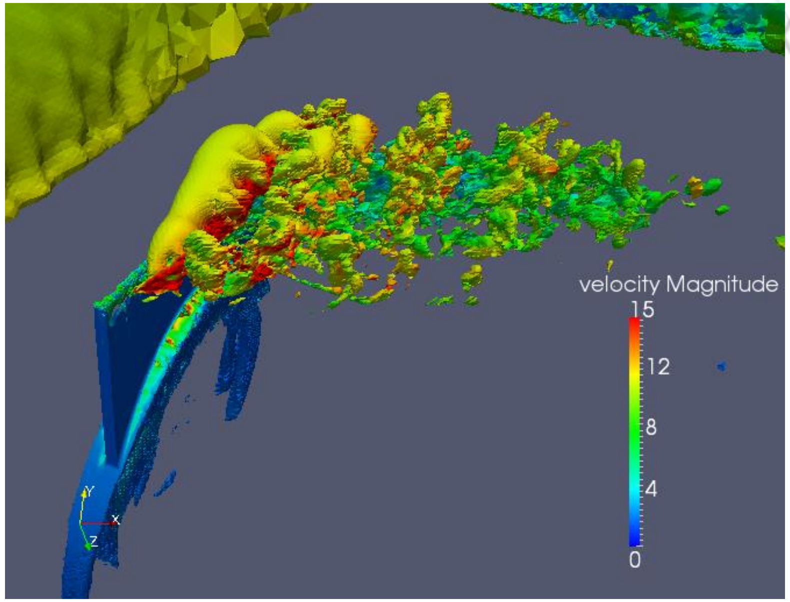
More Catalyst Examples



Code Saturne (EDF)

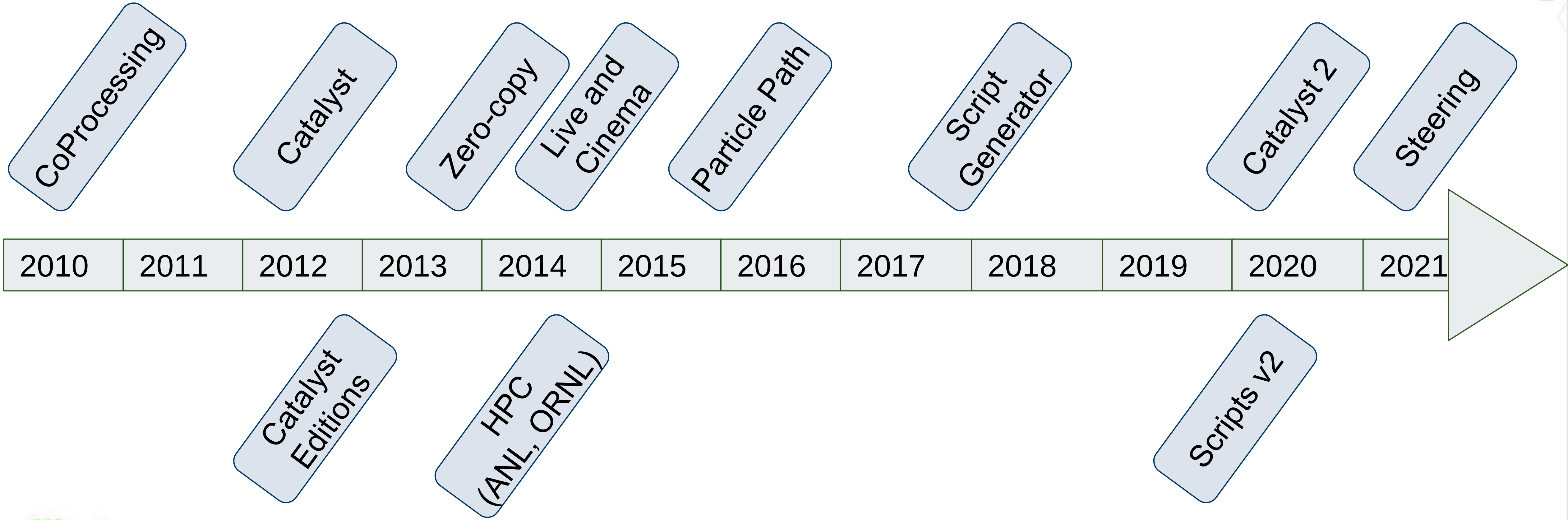


Helios
(Army Aeroflighdynamics
Directorate)



Phasta (UC-Boulder)

Catalyst History



Catalyst 2

Catalyst 2 : challenges

- Make it easy to develop
- Make it easy to build
- Make it easy to deploy
- Make it easy to maintain and upgrade

Catalyst 2 : easy development

- Avoid need to understand VTK data model
- Provide mechanism to provide data with zero-copy & meta-data to interpret it

→ Conduit !

Catalyst 2 : simple API and stable API

https://catalyst-in-situ.readthedocs.io

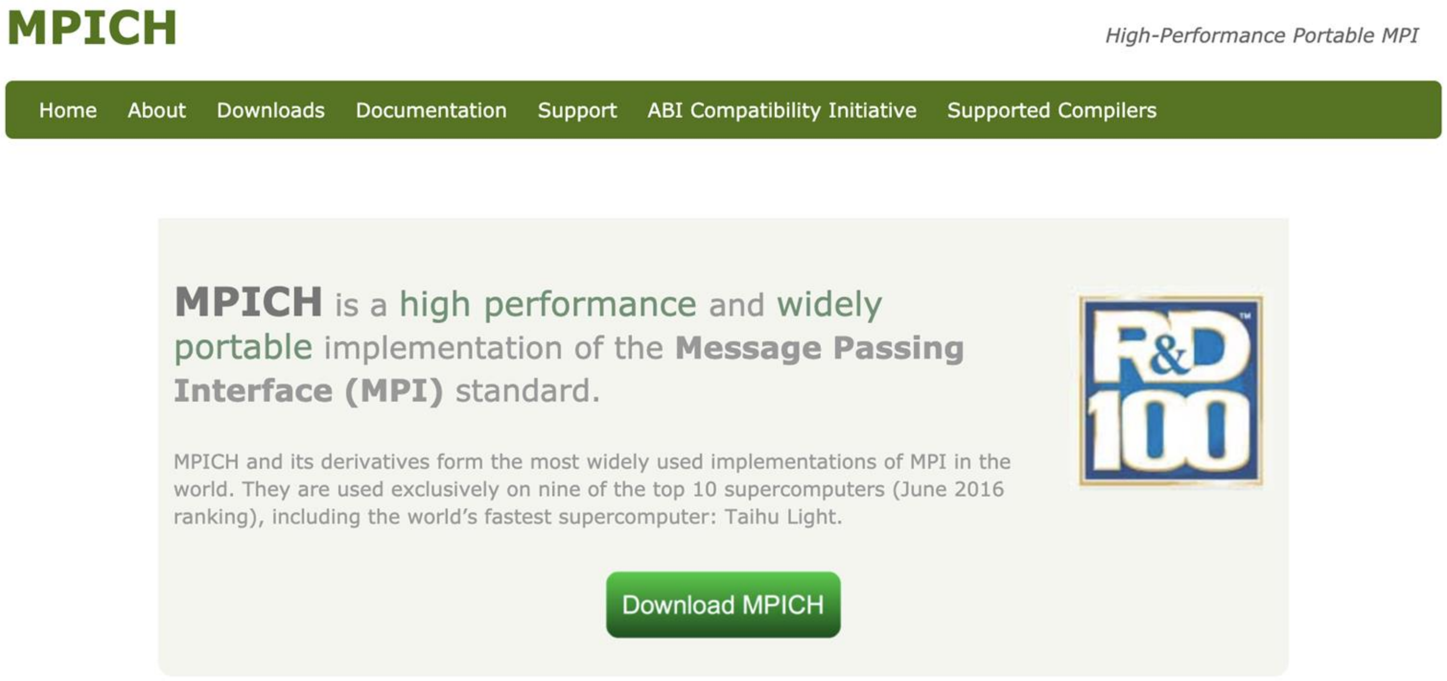
```
enum catalyst_status catalyst_initialize(const conduit_node* params);
enum catalyst_status catalyst_finalize(const conduit_node* params);

enum catalyst_status catalyst_execute(const conduit_node* params);
enum catalyst_status catalyst_results(conduit_node* params);

enum catalyst_status catalyst_about(conduit_node* params);
```

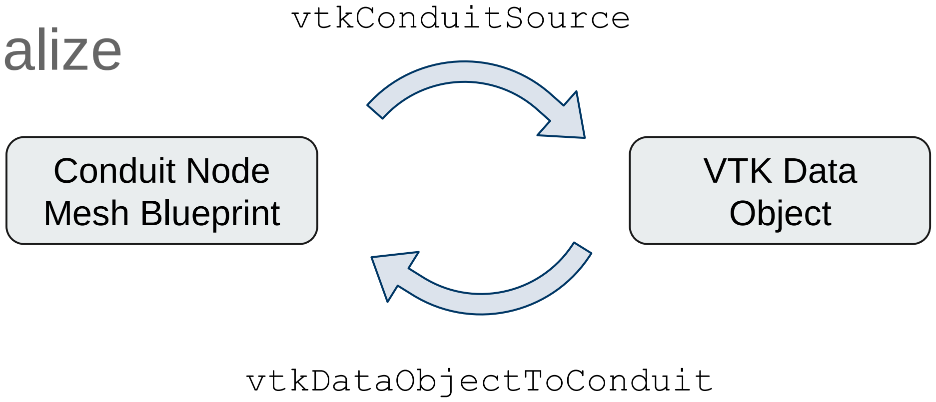
Catalyst 2 API

- Inspired by “MPICH ABI compatibility initiative” (https://www.mpich.org/abi/)



Catalyst 2 implementations

- **Change implementation at run-time**
  - Environment variables
    - CATALYST_IMPLEMENTATION_PATHS
    - CATALYST_IMPLEMENTATION_NAME
  - Dedicated keys in conduit node of catalyst_initialize
    - catalyst_load/search_paths
    - catalyst_load/implementation
  - Legacy way: LD_LIBRARY_PATH

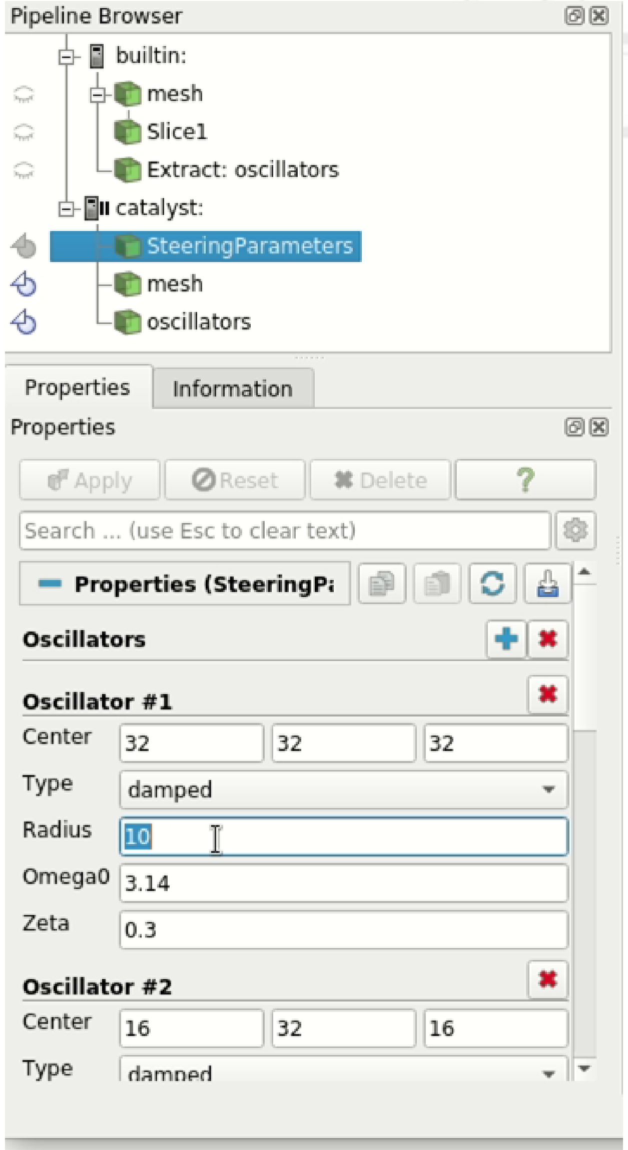


- **Several implementations**
  - catalyst-stub: no external dependencies, suitable for building
  - catalyst-paraview: uses ParaView and VTK to work on data
  - Debug, replay, etc.

Catalyst 2 Steering

Drive the simulation from ParaView via Catalyst API and Live Visualization

```
enum catalyst_status
catalyst_results(conduit_node* params);
```



kitware

Catalyst 2 : goals achieved!

- **Easy to develop**
  - No need to understand VTK
  - No data conversion
- **Easy to deploy**
  - Several catalyst implementations deployment
  - Not link to specific ParaView version
- **Easy to build**
  - Few dependencies using catalyst-stub
  - No CMake
- **Easy to maintain**
  - Stable C API/ABI
  - No need to rebuild simulation for new ParaView version

kitware

Catalyst Road Map

- **Meta data on meshes**
  - Defined in the mesh/state/my_meta_data conduit node
  - Becomes VTK Field Data

- **In-transit analysis with Catalyst**
  - Based on ADIOS

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Conduit + ParaView Catalyst

ParaView Catalyst practical how-to

- Instrumented our simulation code? Yes!
- Compiled with ParaView Catalyst? Yes!

- How do we run and specify visualization tasks????

The ParaView Extractors

- Extractors are items in the visualization pipeline that can save data or images at a user-chosen temporal frequency.
  - Data Extractors
  - Image Extractors

- Using Extractors, no custom code per iteration is really necessary in the majority of cases. One can simply use Extractors to save out images from views or data extracts from filters and other data producers. (*more on that later*)

The ParaView Catalyst Python scripts

- Python scripts, are written by the ParaView application, given a representative template input file, and a set of visualization filters interactively tuned by the user in an *offline* fashion [not connected to a running solver]
- We create a visualization pipeline with the numerous visualization filters and rendering options available in ParaView, and we add “Extractors”
- ParaView provides hybrid parallelism out-of-the-box
  - MPI-enabled
  - TBB multi-threading
  - CUDA-enabled filters
- Some [or most of the] ParaView visualization code will be tightly integrated with the solver memory and execution space.

ParaView Catalyst Python scripts

- Can be generated from an interactive ParaView session, and later fine-tuned
- Are completely interchangeable between the batch-mode ParaView execution (reading data from disk), and the in-situ execution

Catalyst scripts (batch-mode vs. in-situ mode)

```
grid = OpenDataFile(registrationName='grid',
filename=['/LULESH/datasets/data_000009.vtpd'])


renderView1 = GetRenderView()

rep = Show()

ColorBy(rep, ['POINTS', 'velocity'])

Render()


# execute in batch-mode with data read from disk

from paraview.simple import
    SaveExtractsUsingCatalystOptions

SaveExtractsUsingCatalystOptions(options)
```

```
grid = TrivialProducer(registrationName='grid‘)



renderView1 = GetRenderView()

rep = Show()

ColorBy(rep, ['POINTS', 'velocity'])


v = CreateExtractor('VTPD', grid)

v.Trigger = 'TimeStep'

v.Trigger.Frequency = 30

v.Writer.FileName = ‘data_{timestep:06d}.vtpd‘
```

ParaView-Catalyst Blueprint

The Protocol is rather simple:

- Defines the options accepted by **catalyst_initialize()**; these include things like ParaView Python scripts to load, directories to save data

- Defines the protocol for **catalyst_execute()** and includes information about Catalyst channels i.e. ports on which data is made available to in situ processing as well as the actual data from the simulation

- Defines the protocol for **catalyst_finalize()**

ParaView-Catalyst Blueprint

Similar to the Conduit Mesh Blueprint.

```
node["catalyst/scripts/script/filename"] =

node["catalyst/state/cycle"] =
node["catalyst/state/time"] =

node["catalyst/channels/grid/type"] = "mesh"
node["catalyst/channels/grid/data"] =
```

Code instrumentation - before

The Catalyst glue code for the SPH-EXA solver is 144 lines of code

Enabling in-situ visualization can be optionally compiled

```
int main(int argc, char** argv)
{
  MPI_Init_and_Code_Init();

  for (d.iteration = 0; d.iteration <= maxStep; d.iteration++)
  {
  Solve_For_Each_Timestep();


  }


  return exitSuccess();
}
```

Code instrumentation - after

- The Catalyst glue code for the SPH-EXA solver is 144 lines of code
- The execution driver is instrumented with 4 lines of code
- Total: 148 lines of code

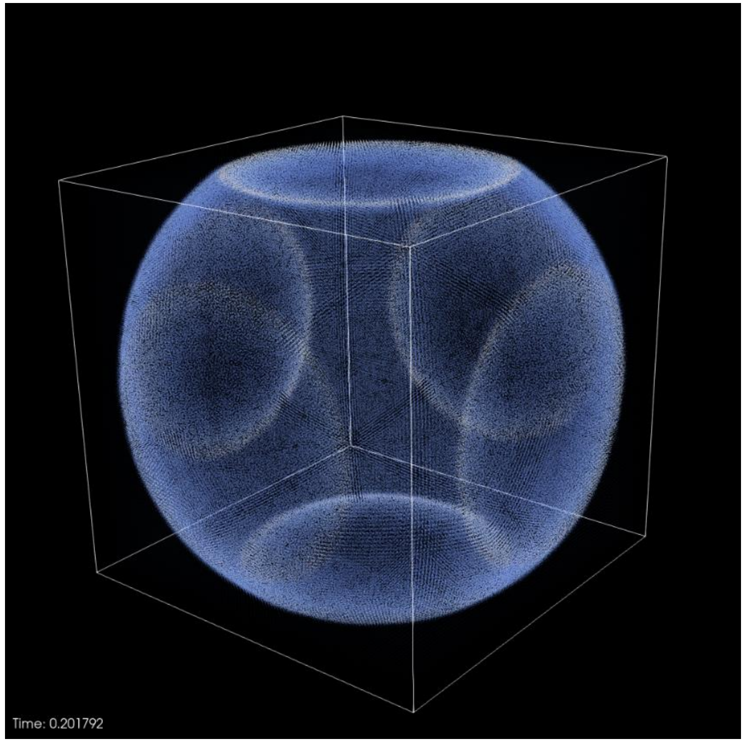
```
#include "CatalystAdaptor.h"
int main(int argc, char** argv)
{
  MPI_Init_and_Code_Init();
  CatalystAdaptor::Initialize(argc, argv);
  for (d.iteration = 0; d.iteration <= maxStep; d.iteration++)
  {
  Solve_For_Each_Timestep();
  CatalystAdaptor::Execute(d, domain.startIndex());
  }
  CatalystAdaptor::Finalize();
  return exitSuccess();
}
```

What about “Operations Control”?

Adaptive Control

- Choose different pathes of execution based on queries/triggers
- The catalyst_execute(ConduitNode) script can be customized

```
def catalyst_execute(node):
  threshold = 1.4
  if reader.PointData[“Density”].GetRange()[1] > threshold:
     print(“density at timestep”, node.timestep)
     # Extract particles where Density > threshold
  else:
     # use ALL particles
```



What about “Operations Control”?

Human in the loop

ParaView has a very efficient client-server mode for post-hoc visualization

- Uses a powerful, remote server to do the heavy work:
  - Parallel I/O
  - Parallel filtering
  - Parallel Image Composition
- Has a very efficient, threshold-based, Image Delivery engine
- The results of the different visualization operations stay on the server (Mbytes, Gbytes, Tbytes (choose your flavour))
- Only the image gets sent over the network to the client (1024x1024 pixels, RGB).

ParaView-Catalyst uses the same engine

- Open a port on a reverse tunnel to my desktop
  - ssh -R 22222:localhost:22222 daint103.cscs.ch

  - Tell my desktop client to send a request for connection on that port
  - my code instrumentation listens to incoming calls with four additional lines of Python code:

from paraview import catalyst
options = catalyst.Options()
options.EnableCatalystLive = 1
options.CatalystLiveURL = 'daint103:22222'

Demonstration of Catalyst Live

Other examples: LULESH instrumented for ParaView Catalyst

[ASC Proxy Apps]

Instrumented with the [VTK-based] Catalyst

Instrumented with the [Conduit-based] Catalyst

Thanks to Utkarsh Ayachit (Kitware)

ParaView pipeline vs. Ascent pipeline

```
renderView1 = CreateView('RenderView')

selection=SelectPoints()
selection.QueryString="Density >= 1.4“

extractSelection = ExtractSelection()
thresholdDisplay = Show(extractSelection)
ColorBy(thresholdDisplay, ['POINTS', 'Density'])

pNG1 = CreateExtractor('PNG', renderView1)
pNG1.Trigger = 'TimeStep'
pNG1.Writer.FileName =
'threshold_{timestep:06d}{camera}.png'
pNG1.Trigger.Frequency = 100
```

```
"action": "add_pipelines",
        "pipelines": {
            "pl1": {
                "f1": {
                    "type": "threshold",
[…]
"action": "add_scenes",
        "scenes": {
            "s1": {
                "plots": {
                    "p1": {
                        "type": "pseudocolor",
                        "pipeline": "pl1",
[…]
        "renders": {
                    "r1": {
                      "image_prefix": "ThresholdImage.%05d",
```

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Third Method. Use Ascent and ParaView Python scripts

Ascent, passing data to a ParaView script

- Ascent Documentation

- The instrumented simulation sends a tree structure (json like) that describes the simulation data using the Conduit Blueprint Mesh specification.
- A ParaView plugin constructs one of the following datasets: vtkImageData, vtkRectilinearGrid, vtkStructuredGrid or vtkUnstructuredGrid
- This data is converted to a VTK format using shallow copies for data arrays

Ascent, passing data to a ParaView script

- Ascent's "action" node is:

```
"action": "add_extracts",
   "extracts":
   {
     "e1":
     {
       "type": "python",
        "params":
        {
        "file": "paraview-vis.py“
        }
     }
  }
```

- The ParaView script describes a "standard" ParaView pipeline and can save images and any other derived data to disk

Summary

- Adopting the two-Conduit-based environments presented is rather simple, very intuitive. The bridges to ParaView or Ascent are now hiding all the complexity from the code developers
- Both environments have proven tracks of high-scale deployments
- What’s next:
  - evaluate performance, memory consumption,
  - visualization pipeline creation and tuning, and re-usability

- For the first time in several decades of developing data formats plugins (converters, adaptors, ..), I don’t have to worry about this anymore, and can concentrate on what I love to do best: !! Visualization !!
- Already 10 years ago, while presenting libSim, people would tell me

“*we’re **very** tight on memory*”

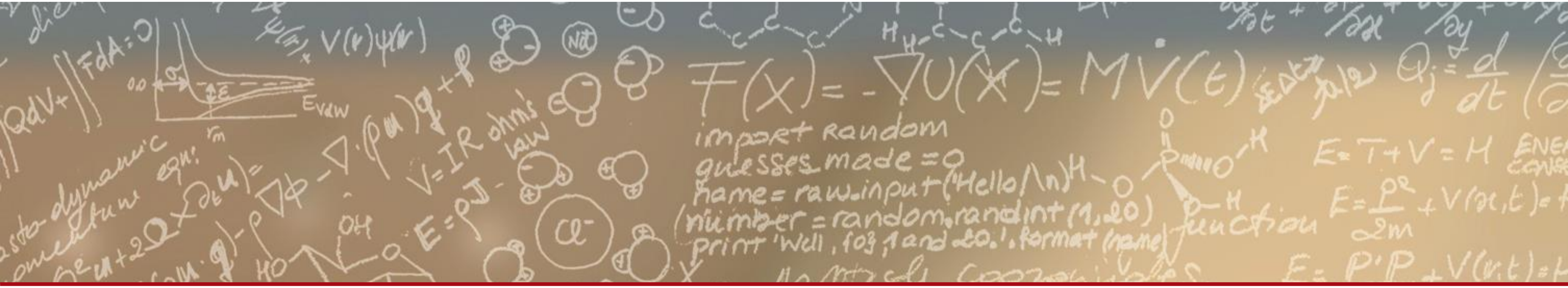
- Adding an in-transit layer is the next venue to explore. Appointment for a CSCS tutorial in 2023.

Resources

- https://github.com/jfavre/InSitu-Vis-Tutorial2022
- My slides
- https://dav.lbl.gov/events/SC21_SENSEI_Ascent_Tutorial/

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Thank you for your attention.